

R.Pan



นักศึกษา
14/03/68

Generating Music Variations through Chaotic Dynamical Systems Exploration

Wannasa Rianthong
Kanatsanun Sub-udom
Patipan Somwong

This report submitted in partial fulfillment of
the requirements for the degree of Bachelor of Sciences in Applied Mathematics
Department of Mathematics and Computer Science

Faculty of Science and Technology
Copyright © 2024 Rajamangala University of Technology Thanyaburi

Generating Music Variations through Chaotic Dynamical Systems Exploration

**Wannasa Rianthong
Kanatsanun Sub-udom
Patipan Somwong**

This report submitted in partial fulfillment of
the requirements for the degree of Bachelor of Sciences in Applied Mathematics
Department of Mathematics and Computer Science
Faculty of Science and Technology
Copyright © 2024 Rajamangala University of Technology Thanyaburi

Title of Project	Generating Music Variations through Chaotic Dynamical Systems Exploration
Student(s)	116410901005-1 Wannasa Rianthong 116410901033-3 Kanatsanun Sub-udom 116410901035-8 Patipan Somwong
Advisor(s)	Dr. Ratthaprom Promkam Akarate Singta

The undersigned project committee of Department of Mathematics and Computer Sciences hereby certifies that this report is submitted to the course 09-115-404 Project in Applied Mathematics and qualified in partial fulfillment of the requirements for the degree of Bachelor of Sciences in Applied Mathematics.

(Asst. Prof. Dr. Wanna Sriprad)

Head of Mathematics Divison

The project committee includes the following.

Advisor

(Dr. Ratthaprom Promkam)

Co-advisor

(Asst. Prof. Dr. Pakeeta Sukprasert)

Co-advisor

(Akarate Singta)

Committee

(Assoc. Prof. Dr. Pongsakorn Sunthrayuth)

Committee

(Assoc. Prof. Dr. Wongvisarut Khuangsatung)

Committee
(Dr. Nonthiya Makate)

Abstract

This work proposes an innovative approach to introducing variation in musical compositions, addressing the challenge of composer burnout. By exploiting the properties of chaotic dynamical systems—renowned for their sensitivity to initial conditions—this method integrates melodic variation with an expanded rhythmic structure. The rhythmic expansion is achieved through controlled prolongation of musical note durations, ensuring a seamless interplay between melodic and rhythmic elements. Furthermore, the implementation of this approach has led to the development of a web application and a Python software library, allowing composers and researchers to comfortably generate, analyze, and experiment with chaotic musical variations in a user-friendly and computationally efficient manner.

Keywords: Chaotic, Dynamical Systems, Music Variation

Mathematics Subject Classification 2020: 37N99, 34H10, 65L06

Acknowledgement

We would like to express our sincere gratitude to Dr. Ratthaprom Promkam, our project advisor, for his invaluable guidance and insightful suggestions. His expertise has been instrumental in demonstrating the feasibility of integrating mathematics with music. Without his support and advice, the completion of this project would not have been possible. His profound knowledge has consistently helped us navigate various challenges throughout the project, making significant contributions to its progress.

Furthermore, we extend our heartfelt appreciation to Asst. Prof. Dr. Pakeeta Sukprasert, our co-advisor, for her continuous support. Her assistance in coordinating with Dr. Ratthaprom Promkam during his absence ensured that we received the necessary guidance at crucial moments. Her unwavering support greatly facilitated our progress and helped us overcome various obstacles.

Lastly, we acknowledge our own perseverance and dedication in bringing this project to fruition. The encouragement and commitment we have shown to ourselves have been essential in achieving our objectives and meeting our expectations.

Authors
3 March 2025

Table of Contents

1	Introduction	1
2	Preliminary	3
2.1	Ordinary Differential Equations	3
2.2	Initial Value Problems	5
2.3	Dynamical Systems	7
2.4	Numerical Solution for solving Initial Value Problems	7
2.4.1	Taylor Series	8
2.4.2	Euler's Method	9
2.4.3	Runge-Kutta Method	10
2.5	Stability Analysis	12
2.6	Chaotic Dynamical Systems	15
2.7	Music Theory	22
2.7.1	Pitch	23
2.7.2	Octaves	23
2.7.3	The Staff and Clefs	24
2.7.4	Duration	24
2.8	Musical Variations	27
3	Main Results	29
3.1	Musical Variations from a Chaotic Mapping Method	29
3.2	Expanded Rhythm Method	36
3.3	Musical Variations from a Expanded Rhythm Method	36
3.4	Showcase	48
4	Discussion and Conclusion	51
4.1	Discussion	51
4.1.1	Python Library	51
4.1.2	Web Application	63
4.1.3	Implementation	68
4.2	Conclusion	79
References		81
Appendix		82
Source Code		83
Full Music Sheet		118

List of Figures

2.1	Example of Euler's Method.	9
2.2	Example of the fourth-order Runge-Kutta method.	12
2.3	Numerical solution of the system (2.9) obtained using the RK4 method.	15
2.4	Bifurcation diagram of the Lorenz system for ρ ranging from 0 to 40.	18
2.5	Bifurcation diagram of the Lorenz system for σ ranging from 0 to 20.	19
2.6	Bifurcation diagram of the Lorenz system for β ranging from 0 to 5.	20
2.7	The Lorenz system with the initial condition (1, 1, 1) and parameters $\rho = 22$, $\rho = 23$, $\rho = 24$, and $\rho = 25$	21
2.8	The original and variation example.	28
3.1	Mapping α	34
3.2	Mapping β	34
3.3	The original and new variation of Ah vous dirai-je Maman.	34
3.4	The figure for visualizes how a chaotic mapping method can be used to generate musical variations.	35
3.5	The original and musical variation of Ah vous dirai-je Maman.	36
3.6	Mapping process of α	43
3.7	Mapping process of $\overset{\rightarrow}{\beta}$	43
3.8	The original and musical variation from a expanded rhythm method of Ah vous dirai-je Maman.	44
3.9	The figure for visualizes how a chaotic mapping with musical variation method can be used to generate musical variations	45
3.10	The original and musical variation from a expanded rhythm method of Ah vous dirai-je Maman with mapping $\overset{\leftarrow}{\beta}$	46
3.11	Visualizes the result of using the mapping $\overset{\leftarrow}{\beta}$	47
3.12	First two bars of Pachelbel's Canon.	49
3.13	New variation of Pachelbel's Canon using the mapping $\overset{\rightarrow}{\beta}$ in the first two bars. .	49
3.14	New variation of Pachelbel's Canon using the mapping $\overset{\leftarrow}{\beta}$ in the first two bars. .	49
3.15	First two bars of River Flows In You.	50
3.16	New variation of River Flows In You using the mapping $\overset{\rightarrow}{\beta}$ in the first two bars.	50
3.17	New variation of River Flows In You using the mapping $\overset{\leftarrow}{\beta}$ in the first two bars.	50
4.1	Web application interface.	65
4.2	Graphical representation of the interface displaying the Sidebar.	74
4.3	Graphical representation of the interface displaying the results webpage.	77
4.4	Graphical representation of the interface displaying the settings webpage.	79

List of Tables

2.1	First five numerical solutions of the system (2.9) computed using the RK4 method.	15
3.1	The result of a mapping g from $\{\phi_1(kh)\}_{k=0}^{10}$ to $\{p_k\}_{k=0}^{10}$ and the result of a mapping l from $\{\tilde{\phi}_1(kh)\}_{k=0}^{10}$ to new musical pitches.	33
3.2	The result of a mapping g from $\{\phi_1(kh)\}_{k=0}^{13}$ to $\{p_k\}_{k=0}^{13}$ and the result of a mapping l from $\{\tilde{\phi}_1(kh)\}_{k=0}^{13}$ to new expanded musical pitches.	42

Chapter 1

Introduction

Music variation serves as a catalyst for creative thinking in the songwriting process. It offers flexibility, capable of generating patterns ranging from close replicas to entirely different ones. The outcome depends on the composer's desires. When applied to compositions, it's like creating another version of the same song, making the music open to change every time it's heard. In the past, composers often employed techniques like inversion, retrograde or sections of music to expand upon the original musical content. However, these techniques gradually lost their appeal and were seen as tiresome. Music variation steps in to fill this gap. This technique opens up possibilities for composers to create entirely new musical patterns without being tied to the original framework. Individuals can transform the written notes into their own unique dynamic and fresh music.

Nowadays, artificial intelligence (AI) technologies have significantly advanced, enabling them to create music with ever-increasing proficiency [1]. Well-known AI music generation platforms such as Mubert [2] and Musicity [3] empower users with real-time music generation capabilities, enabling them to effortlessly select their preferred genre or mood and promptly receive a personalized soundtrack tailored to their preferences. On the other hand, Soundraw [4] and Boomy [5] function as AI-driven music creation tools, furnishing a diverse array of features to aid users in sculpting their musical opuses with ease. Meanwhile, AIVA [6] harnesses the power of deep learning to craft original music closely resembling the distinctive style of a particular artist or genre. Users can furnish reference tracks or articulate their desired musical aesthetics, prompting AIVA to generate fresh compositions that align precisely with their specifications. Even though AI provides these capabilities, it often requires high computational resources, making it unable to run on devices with low processing power. Additionally, some AI music composition tools may produce music in limited styles.

Since limitations of AI music technology is an expensive problem to leave unaddressed, the following consequences it may lead to. Aspiring artists and musicians will miss out on the opportunity to use these tools due to limited access, as most people lack high-performance equipment. Furthermore, all music generated by AI may start to sound similar, potentially leading to a lack of musical diversity.

This work focuses on three main objectives. Firstly, developing an alternative algorithm that uses fewer resources for generating music through a chaotic dynamical system. Secondly, demonstrating that chaotic dynamical systems can be applied to create new music based on existing compositions. Lastly, providing composers with limited resources and tools the opportunity to utilize the algorithm as a source of inspiration for further songwriting. To achieve these goals, this study begins with melodic variation with expanded rhythm, which is then translated into numerical values. These numerical values are then input into a chaotic dynamical system, resulting in a new set of numbers different from the original. As a result, by mapping these numbers back to musical notes, leading to the creation of a new piece of music. The algorithm developed through this process was further enhanced, resulting in a web application and a Python library. This method requires lower computing resources compared to using AI music composition technology and enables the creation of diverse musical compositions depending on the original song, initial values, and equations used.

The structure of this paper is organized as follows. Chapter 2 provides an overview of essential mathematical notations and fundamental music theory concepts relevant to our study.

We explore the use of chaotic dynamical systems to generate musical variations. Moreover, toward the end of this chapter, an alternative method utilizing melodic variation with expanded rhythm is introduced, which further enriches the diversity of musical compositions presented in Chapter 3. Finally, Chapter 4 focuses on the development of a Python library and a web application, detailing their functionality, installation process, and applications for both researchers and musicians.

Chapter 2

Preliminary

In this chapter, we present the mathematical notations and content used in our project to provide a clear understanding of its foundation.

Throughout this project, we use the following notation. The set of real numbers is denoted by \mathbb{R} , while the set of nonnegative real numbers is denoted by $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geq 0\}$. Similarly, the set of natural numbers is denoted by \mathbb{N} . For any fixed $n \in \mathbb{N}$, we define $\mathbb{N}_n := \{0, 1, 2, \dots, n\}$, representing the first n natural numbers. A sequence of real numbers is written as $\{a_k\}_{k=0}^n = \{a_0, a_1, \dots, a_n\}$, where each $a_k \in \mathbb{R}$ for $k = 0, 1, \dots, n$. The Cartesian product of \mathbb{R} with itself is denoted by $\mathbb{R} \times \mathbb{R}$ and is defined as

$$\mathbb{R} \times \mathbb{R} = \{(x, y) \mid x, y \in \mathbb{R}\}.$$

More generally, the n -dimensional Cartesian product of \mathbb{R} , denoted by \mathbb{R}^n , is defined as

$$\mathbb{R}^n = \underbrace{\mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R}}_{n \text{ times}} = \{(x_1, x_2, \dots, x_n) \mid x_1, x_2, \dots, x_n \in \mathbb{R}\}.$$

The norm of a vector $v = (v_1, v_2, \dots, v_n) \in \mathbb{R}^n$ is denoted by $\|v\|$ and is given by

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}.$$

A function f mapping from \mathbb{R} to \mathbb{R} is denoted as $f : \mathbb{R} \rightarrow \mathbb{R}$, with its domain represented by $\text{dom}(f)$. A function f is said to be one-to-one (injective) if

$$f(x_1) = f(x_2) \implies x_1 = x_2.$$

2.1 Ordinary Differential Equations

In this section, we focus on systems of ordinary differential equations and demonstrate that any ordinary differential equation can be transformed into a system of first-order ordinary differential equations.

A system of ordinary differential equations is a system of first-order differential equations of the form:

$$\begin{aligned} x'_1(t) &= f_1(t, x_1(t), x_2(t), \dots, x_n(t)), \\ x'_2(t) &= f_2(t, x_1(t), x_2(t), \dots, x_n(t)), \\ &\vdots \\ x'_n(t) &= f_n(t, x_1(t), x_2(t), \dots, x_n(t)), \end{aligned} \tag{2.1}$$

where t is the independent variable defined on an interval I , $(x_1(t), x_2(t), \dots, x_n(t)) \in \mathbb{R}^n$ are the dependent variables, and f_i are functions from $I \times \mathbb{R}^n$ to \mathbb{R} for $i = 1, 2, \dots, n$. The notation $x'_1(t), x'_2(t), \dots, x'_n(t)$ represents the first-order derivatives of $x_1(t), x_2(t), \dots, x_n(t)$ with respect to t .

Any higher-order ordinary differential equation can be rewritten as a system of first-order ordinary differential equations. Consider a general n th-order ordinary differential equation:

$$x^{(n)}(t) = f(t, x(t), x'(t), x''(t), \dots, x^{(n-1)}(t)).$$

By introducing new variables:

$$x_1(t) = x(t), \quad x_2(t) = x'(t), \quad x_3(t) = x''(t), \quad \dots, \quad x_n(t) = x^{(n-1)}(t),$$

we obtain the equivalent first-order system:

$$\begin{aligned} x'_1(t) &= x_2(t), \\ x'_2(t) &= x_3(t), \\ &\vdots \\ x'_{n-1}(t) &= x_n(t), \\ x'_n(t) &= f(t, x_1(t), x_2(t), \dots, x_n(t)). \end{aligned}$$

This transformation allows us to analyze higher-order differential equations using the theory and methods developed for systems of first-order ordinary differential equations. Additionally, we can express a system of differential equations in the form:

$$\dot{x}(t) = f(t, x(t)), \tag{2.2}$$

where $x(t) \in \mathbb{R}^n$. The terms $\dot{x}(t)$ and $f(t, x(t))$ can be represented in vector notation as follows:

$$\dot{x}(t) = \begin{bmatrix} x'_1(t) \\ x'_2(t) \\ \vdots \\ x'_n(t) \end{bmatrix}, \quad \text{and} \quad f(t, x(t)) = \begin{bmatrix} f_1(t, x_1(t), x_2(t), \dots, x_n(t)) \\ f_2(t, x_1(t), x_2(t), \dots, x_n(t)) \\ \vdots \\ f_n(t, x_1(t), x_2(t), \dots, x_n(t)) \end{bmatrix}.$$

This formulation demonstrates that a system of ordinary differential equations can be expressed in vector form, facilitating a more compact and systematic representation.

Definition 2.1 ([7]). A function $\varphi : I \rightarrow \mathbb{R}^n$ is called a solution of the system (2.2) if the following conditions hold:

1. φ is continuously differentiable on I .
2. For every $t \in I$, $\varphi(t)$ satisfies $\dot{\varphi}(t) = f(t, \varphi(t))$.

Example 2.2. Consider the following system of ordinary differential equations:

$$x'_1(t) = x_2(t), \tag{2.3}$$

$$x'_2(t) = -x_1(t). \tag{2.4}$$

We will verify that the functions

$$\varphi_1(t) = \cos t + \sin t, \tag{2.5}$$

$$\varphi_2(t) = -\sin t + \cos t \tag{2.6}$$

constitute a solution of the system (2.3)-(2.4) according to Definition 2.1. The functions φ_1 and φ_2 are sums of sine and cosine functions, which are continuously differentiable on all of \mathbb{R} . Differentiating (2.5):

$$\dot{\varphi}_1(t) = -\sin t + \cos t.$$

Differentiating (2.6):

$$\dot{\varphi}_2(t) = -\cos t - \sin t.$$

Now, substituting $\varphi_1(t)$ and $\varphi_2(t)$ into equation (2.3):

$$\begin{aligned}\dot{\varphi}_1(t) &= \varphi_2(t), \\ -\sin t + \cos t &= -\sin t + \cos t.\end{aligned}$$

which confirms equation (2.3) holds. Similarly, substituting $\varphi_2(t)$ and $\varphi_1(t)$ into equation (2.4):

$$\begin{aligned}\dot{\varphi}_2(t) &= -\varphi_1(t), \\ -\cos t - \sin t &= -(\cos t + \sin t), \\ -\cos t - \sin t &= -\cos t - \sin t.\end{aligned}$$

which confirms equation (2.4) holds.

Since both conditions of Definition 2.1 are satisfied, we conclude that $\varphi_1(t) = \cos t + \sin t$ and $\varphi_2(t) = -\sin t + \cos t$ form a solution of the given system on \mathbb{R} .

2.2 Initial Value Problems

An initial value problem is a system of first-order differential equations of the form:

$$\dot{x}(t) = f(t, x(t)), \quad x(t_0) = x_0, \quad (2.7)$$

where t is the independent variable defined on an interval I , $x(t) \in \mathbb{R}^n$ represents state of the system at time t , $\dot{x}(t)$ is its derivative with respect to the independent variable t , and $f : I \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a given function. Additionally, $x(t_0)$ is an initial condition at time t_0 .

Definition 2.3 ([7]). A function $\varphi : I \rightarrow \mathbb{R}^n$ is called a solution of the system (2.7) if the following conditions hold:

1. φ is continuously differentiable on I .
2. For every $t \in I$, $\varphi(t)$ satisfies $\dot{\varphi}(t) = f(t, \varphi(t))$.
3. The function φ satisfies the initial condition: $\varphi(t_0) = x_0$.

Example 2.4. Consider the initial value problem:

$$\dot{x}(t) = \alpha x(t), \quad x(t_0) = x_0,$$

where $\alpha > 0$ is a parameter. We will verify that the function

$$\varphi(t) = x_0 e^{\alpha(t-t_0)}$$

is a solution to this initial value problem according to Definition 2.3. The function $\varphi(t)$ is an exponential function, which is continuously differentiable on all of \mathbb{R} . Differentiating $\varphi(t)$:

$$\begin{aligned}\dot{\varphi}(t) &= \frac{d}{dt} (x_0 e^{\alpha(t-t_0)}) \\ &= x_0 \cdot \alpha e^{\alpha(t-t_0)} \\ &= \alpha x_0 e^{\alpha(t-t_0)} \\ &= \alpha \varphi(t).\end{aligned}$$

Thus, $\varphi(t)$ satisfies the given differential equation. To verify the initial condition:

$$\begin{aligned}\varphi(t_0) &= x_0 e^{\alpha(t_0-t_0)} \\ &= x_0 e^0 \\ &= x_0.\end{aligned}$$

Since all conditions of Definition 2.3 are satisfied, we conclude that $\varphi(t) = x_0 e^{\alpha(t-t_0)}$ is the solution to the initial value problem.

From the example, we have demonstrated that $x(t)$ satisfies both the differential equation and the initial condition, confirming it as a solution to the initial value problem. However, this does not guarantee that every initial value problem always has a solution or that the solution we obtained is unique. In the next content of initial value problem, we will explore the conditions under which an initial value problem has a solution and whether that solution is unique, addressing the existence and uniqueness of solutions.

Definition 2.5 (Lipschitz Continuity [8]). A function

$$f : D \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^m$$

is said to be **Lipschitz continuous** on D if there exist a constant $L \geq 0$ such that

$$\|f(x_1) - f(x_2)\| \leq L\|x_1 - x_2\| \text{ for all } x_1, x_2 \in D.$$

Example 2.6. Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be given by

$$f(x) = ax + b,$$

where $a, b \in \mathbb{R}$. We will show that f is Lipschitz continuous. For any $x_1, x_2 \in \mathbb{R}$,

$$|f(x_1) - f(x_2)| = |(ax_1 + b) - (ax_2 + b)| = |a(x_1 - x_2)| = |a| \cdot |x_1 - x_2|.$$

Thus,

$$|f(x_1) - f(x_2)| \leq |a| \cdot |x_1 - x_2|.$$

Hence f is Lipschitz continuous with constant $L = |a|$.

Theorem 2.7 (Picard–Lindelöf Existence and Uniqueness [8]). *Let $I = [a, b] \subset \mathbb{R}$ be an interval and $f : I \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ be a continuous function. If f is **Lipschitz continuous** in the state variable x , then for any $t_0 \in I$ and any initial condition $x_0 \in \mathbb{R}^n$, the initial value problem*

$$\dot{x}(t) = f(t, x(t)), \quad x(t_0) = x_0,$$

has a unique solution $x(t)$ on some subinterval $[t_0 - \delta, t_0 + \delta] \subset I$.

Example 2.8. Consider the initial value problem:

$$\dot{x}(t) = x(t), \quad x(0) = 1.$$

We will apply the Picard–Lindelöf theorem to establish existence and uniqueness of the solution.

First, define $f(t, x(t)) = x(t)$. The function f is continuous in both t and x over any domain in \mathbb{R} .

Next, we check the Lipschitz condition in x :

$$|f(t, x_1(t)) - f(t, x_2(t))| = |x_1(t) - x_2(t)|.$$

Since $|x_1(t) - x_2(t)| \leq L|x_1(t) - x_2(t)|$ with $L = 1$, we conclude that f is Lipschitz continuous with constant $L = 1$. By the Picard–Lindelöf theorem, the initial value problem has a unique solution on some interval around $t = 0$.

2.3 Dynamical Systems

In this project, we focus on continuous dynamical systems only, as we will utilize chaotic systems, which are a type of continuous dynamical system discussed in the chaotic systems section.

Definition 2.9. A continuous dynamical system is represented by a set of ordinary differential equations:

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

where $x(t) \in \mathbb{R}^n$ and $f : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$.

Example 2.10 (Lotka-Volterra Equations). The Lotka-Volterra equations describe a continuous dynamical system in the form:

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

where

$$\dot{x}(t) = \begin{bmatrix} x'_1(t) \\ x'_2(t) \end{bmatrix}, \quad f(t, x(t)) = \begin{bmatrix} ax_1(t) - bx_1(t)x_2(t) \\ cx_1(t)x_2(t) - dx_2(t) \end{bmatrix}.$$

Here, $x_1(t)$ represents the prey population, while $x_2(t)$ denotes the predator population. The system parameters are defined as follows:

- $a > 0$: growth rate of prey in the absence of predators,
- $b > 0$: predation rate coefficient,
- $c > 0$: growth rate of predators due to prey consumption,
- $d > 0$: natural mortality rate of predators.

A continuous dynamical system generally includes the concepts of solution definition, existence, and uniqueness, similar to an initial value problem. Therefore, in this section, we have introduced only the definition of a continuous dynamical system.

In the next section, we will discuss methods for finding solutions to continuous dynamical systems, provided a solution exists.

2.4 Numerical Solution for solving Initial Value Problems

In various scientific and technological fields, differential equations serve as essential tools for modeling complex systems. However, finding analytical solutions is often limited to specific cases, making them impractical for many real-world scenarios. In such cases, numerical methods provide a crucial approach to approximating solutions when analytical solutions are difficult or impossible to obtain. The accuracy of numerical methods depends on various factors, such as the chosen method, step size and error accumulation, with some techniques yielding more precise approximations than others. Ultimately, numerical methods play a significant role in solving problems related to differential equations, enabling the determination of solutions for complex systems that cannot be resolved analytically.

This section focuses on the application of numerical methods to solve initial value problems, which involve differential equations with specified initial conditions, where the solution describes the system's behavior over time. Numerical techniques such as Euler's Method and Runge-Kutta Methods are employed to approximate the solution with sufficient accuracy.

2.4.1 Taylor Series

Definition 2.11 (Taylor Series [9]). The Taylor series of a real-valued function $f(x)$, that is infinitely differentiable at $x = a$, is given by:

$$f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \frac{f'''(a)}{3!}(x - a)^3 + \dots$$

or in summation form,

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x - a)^n.$$

Theorem 2.12 (Taylor Series Convergence Theorem). *Let $f(x)$ be an infinitely differentiable function on an open interval I containing a . The Taylor series of $f(x)$ centered at a converges to $f(x)$ for all x in I if and only if the remainder term:*

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - a)^{n+1}, \quad \text{for some } \xi \in (a, x)$$

satisfies:

$$\lim_{n \rightarrow \infty} R_n(x) = 0.$$

Example 2.13 (Exponential Function). Find the Taylor series of $f(x) = e^x$ around $x = 0$.

Solution: The Taylor series is given by:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n.$$

Compute derivatives:

$$\begin{aligned} f(x) &= e^x, \\ f'(x) &= e^x, \\ f''(x) &= e^x, \\ f'''(x) &= e^x, \dots \end{aligned}$$

Since $f^{(n)}(0) = e^0 = 1$, we substitute:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Example 2.14 (Sine Function). Find the Taylor series of $f(x) = \sin x$ around $x = 0$.

Solution: Compute derivatives:

$$\begin{aligned} f(x) &= \sin x, \\ f'(x) &= \cos x, \\ f''(x) &= -\sin x, \\ f'''(x) &= -\cos x. \end{aligned}$$

Evaluating at $x = 0$:

$$f(0) = 0, \quad f'(0) = 1, \quad f''(0) = 0, \quad f'''(0) = -1, \quad f''''(0) = 0, \dots$$

Only odd powers of x remain, leading to:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

2.4.2 Euler's Method

Euler's Method is a fundamental numerical technique that approximates solutions iteratively using a first-order linear approximation of the derivative in the context of Initial Value Problems. Despite its computational simplicity and ease of implementation, the method exhibits relatively low accuracy and is susceptible to stability issues, particularly when employing larger step sizes.

Define h to be the time step size and $t_i = t_0 + ih$. The goal is to construct a discrete approximation $\{x_n\}$ to $x(t)$ at times $t_n = t_0 + nh$, where x_n represents the value of x at t_n .

Euler's Method formula using a first-order Taylor expansion,

$$x(t+h) \approx x(t) + hf(t, x(t)).$$

Truncating after the first derivative term gives the Euler update formula,

$$x_{n+1} = x_n + hf(t_n, x_n).$$

Example 2.15 (Exponential Growth). Solve numerically using Euler's method,

$$\dot{x} = x, \quad x(0) = 1, \quad h = 0.1.$$

Using discrete time steps given by,

$$t_n = t_0 + nh.$$

Where $t_0 = 0$ and a fixed step size h is used, we obtain the discrete time points,

$$t_0 = 0, \quad t_1 = h, \quad t_2 = 2h, \quad t_3 = 3h, \quad \dots, \quad t_n = nh.$$

Using Euler's method, the numerical approximation is given by

$$x_{n+1} = x_n + hx_n.$$

Starting at $x(t_0) = x_0 = 1$,

$$\begin{aligned} x(t_1) &= x_1 = 1 + 0.1(1) = 1.1, \\ x(t_2) &= x_2 = 1.1 + 0.1(1.1) = 1.21, \\ x(t_3) &= x_3 = 1.21 + 0.1(1.21) = 1.331. \end{aligned}$$

Compare with exact solution $x(t) = e^t$ as shown in Figure 2.1.

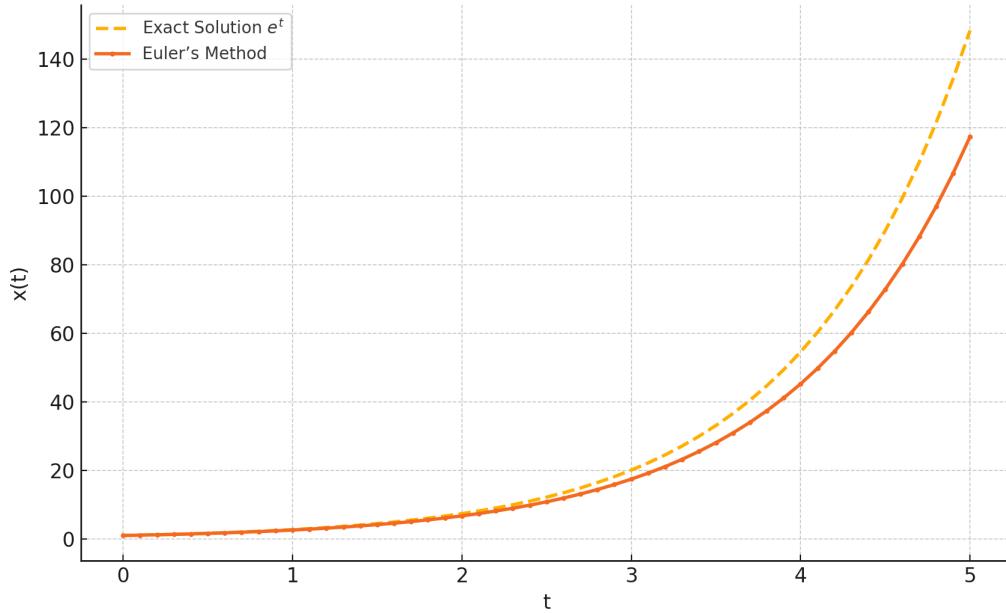


Figure 2.1: Example of Euler's Method.

Euler's method slightly underestimates the true solution because it approximates the curve using linear steps, while the exact solution grows exponentially.

2.4.3 Runge-Kutta Method

The Taylor's series method of solving differential equations numerically is restricted by the labor involved in finding the higher order derivatives. However, there is a class of methods known as Runge-Kutta methods which do not require the calculations of higher order derivatives and give greater accuracy. The Runge-Kutta formulae possess the advantage of requiring only the function values at some selected points solution of the initial value problems. Among these methods, the fourth-order Runge-Kutta method, represented by RK4, is the most commonly used due to its balance between computational efficiency and accuracy. In summary, the Runge-Kutta method is an efficient approach for solving differential equations numerically, as it reduces the complexity of computing higher-order derivatives while maintaining higher accuracy compared to Euler's method.

Consider the problem (2.7). The formula for the fourth-order Runge-Kutta method is given below.

$$\begin{aligned}x_0 &= x_n, \\k_1 &= hf(t_n, x_n), \\k_2 &= hf\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}\right), \\k_3 &= hf\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}\right), \\k_4 &= hf(t_n + h, x_n + k_3), \\x_{n+1} &= x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).\end{aligned}$$

Here, h is the step size, x_n is the approximation of $x(t_n)$ at step n , and $f(t, x)$ represents the given differential equation. The initial condition is given by $x(t_0) = x_0$. This method computes an approximate solution, meaning that $x_n \approx x(t_n)$, providing a higher accuracy numerical solution.

Example 2.16 (Fourth-order Runge-Kutta method). Solve numerically using the fourth-order Runge-Kutta method,

$$\dot{x} = x, \quad x(0) = 1.$$

Using a fixed step size of $h = 0.1$, we define discrete time steps as,

$$t_n = t_0 + nh.$$

where $t_0 = 0$, and we obtain the discrete time points,

$$t_0 = 0, \quad t_1 = 0.1, \quad t_2 = 0.2, \quad t_3 = 0.3, \quad t_4 = 0.4.$$

Using the RK4 method with a step size of $h = 0.1$, we consider the initial condition.

$$t_0 = 0, x_0 = 1.$$

$$t_1 = 0.1$$

$$\begin{aligned}k_1 &= 0.1f(0, 1) = 0.1, \\k_2 &= 0.1f(0.05, 1.05) = 0.105, \\k_3 &= 0.1f(0.05, 1.0525) = 0.10525, \\k_4 &= 0.1f(0.1, 1.10525) = 0.110525, \\x_1 &= x_0 + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} = 1.105171.\end{aligned}$$

$$t_2 = 0.2$$

$$\begin{aligned} k_1 &= 0.1f(0.1, 1.105171) = 0.1105171, \\ k_2 &= 0.1f(0.15, 1.16042955) = 0.116042955, \\ k_3 &= 0.1f(0.15, 1.16319248) = 0.116319248, \\ k_4 &= 0.1f(0.2, 1.22149025) = 0.122149025, \\ x_2 &= x_1 + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} = 1.221403. \end{aligned}$$

$$t_3 = 0.3$$

$$\begin{aligned} k_1 &= 0.1f(0.2, 1.221403) = 0.1221403, \\ k_2 &= 0.1f(0.25, 1.28247315) = 0.128247315, \\ k_3 &= 0.1f(0.25, 1.28552666) = 0.128552666, \\ k_4 &= 0.1f(0.3, 1.34995566) = 0.134995566, \\ x_3 &= x_2 + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} = 1.349858. \end{aligned}$$

$$t_4 = 0.4$$

$$\begin{aligned} k_1 &= 0.1f(0.3, 1.349858) = 0.1349858, \\ k_2 &= 0.1f(0.35, 1.4173509) = 0.14173509, \\ k_3 &= 0.1f(0.35, 1.42072555) = 0.142072555, \\ k_4 &= 0.1f(0.4, 1.49193055) = 0.149193055, \\ x_4 &= x_3 + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} = 1.491824. \end{aligned}$$

Starting at $x_0 = 1$, we compute

$$\begin{aligned} x_1 &= 1.105171, \\ x_2 &= 1.221403, \\ x_3 &= 1.349858, \\ x_4 &= 1.491824. \end{aligned}$$

Compare with the exact solution $x(t) = e^t$ as shown in Figure 2.2.

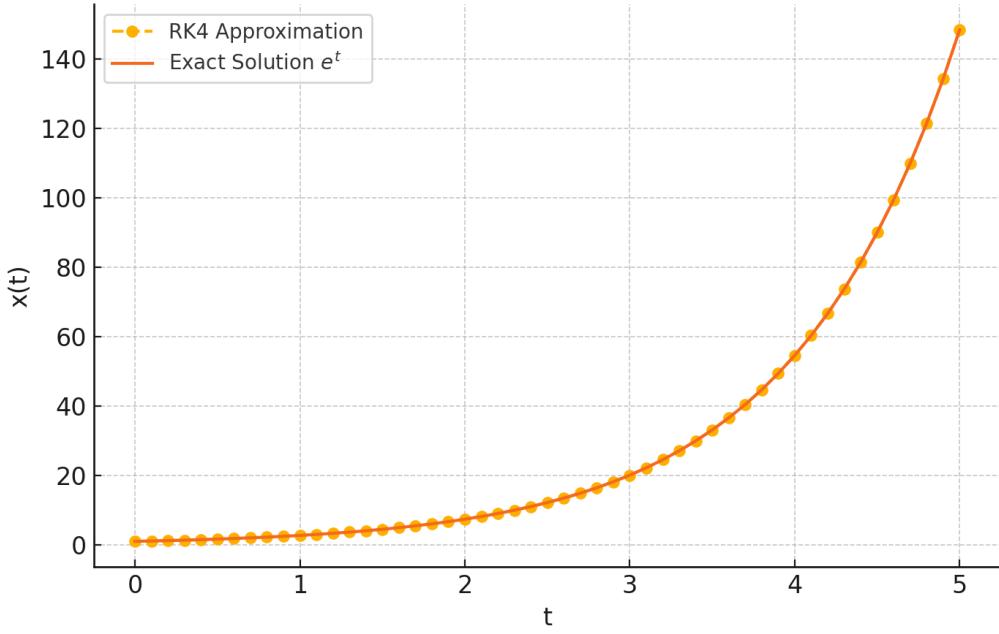


Figure 2.2: Example of the fourth-order Runge-Kutta method.

RK4 provides a highly accurate numerical approximation compared to the exact solution, demonstrating its effectiveness over simpler methods like Euler's method.

2.5 Stability Analysis

The stability of dynamical systems is a fundamental concept in understanding the behavior of systems over time. It can be analyzed using differential equations, which describe the evolution of a system. A general form of a dynamical system is given by:

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

where $x(t) \in \mathbb{R}^n$ and $f : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$.

Equilibrium points (or fixed points) are states where the system remains unchanged over time. These points satisfy:

$$f(t, x(t)) = 0.$$

Example 2.17. Consider the nonlinear differential equation:

$$\frac{dx}{dt} = x(1 - x)$$

To find the equilibrium points, we solve for x where $f(x) = 0$:

$$x(1 - x) = 0$$

which gives the solutions:

$$x = 0 \quad \text{or} \quad x = 1.$$

Thus, the system has two equilibrium points at $x = 0$ and $x = 1$.

Definition 2.18 (Jacobian Matrix). Let $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ have components

$$f(x) = (f_1(x), f_2(x), \dots, f_m(x)).$$

If all partial derivatives exist and are continuous in an open region $D \subset \mathbb{R}^n$, then the **Jacobian matrix** of f at $x = (x_1, x_2, \dots, x_n) \in D$ is

$$\frac{\partial f}{\partial x}(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \cdots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x) & \frac{\partial f_m}{\partial x_2}(x) & \cdots & \frac{\partial f_m}{\partial x_n}(x) \end{bmatrix}. \quad (2.8)$$

Example 2.19. Define

$$f(x_1, x_2) = (x_1^2 + x_2^2, x_1 + x_2^3).$$

Then $f_1(x_1, x_2) = x_1^2 + x_2^2$ and $f_2(x_1, x_2) = x_1 + x_2^3$. The Jacobian is,

$$\frac{\partial f}{\partial x}(x_1, x_2) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x_1, x_2) & \frac{\partial f_1}{\partial x_2}(x_1, x_2) \\ \frac{\partial f_2}{\partial x_1}(x_1, x_2) & \frac{\partial f_2}{\partial x_2}(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 2x_1 & 2x_2 \\ 1 & 3x_2^2 \end{bmatrix}.$$

The stability of an equilibrium point depends on how the system behaves when slightly perturbed from it. To determine the stability of an equilibrium point, we analyze the Jacobian matrix (2.8).

The eigenvalues of A indicate the nature of the equilibrium. If all eigenvalues have negative real parts, the equilibrium is stable, meaning small perturbations will decay, and the system will return to equilibrium. Conversely, if at least one eigenvalue has a positive real part, the equilibrium is unstable, meaning small perturbations will grow, leading the system away from equilibrium.

Definition 2.20 ([10]). If A is an $n \times n$ matrix, then a nonzero vector \mathbf{x} in \mathbb{R}^n is called an *eigenvector* of A if $A\mathbf{x}$ is a scalar multiple of \mathbf{x} ; that is,

$$A\mathbf{x} = \lambda\mathbf{x}$$

for some scalar λ . The scalar λ is called an *eigenvalue* of A , and \mathbf{x} is said to be an *eigenvector corresponding to λ* .

Theorem 2.21 ([10]). *If A is an $n \times n$ matrix, then λ is an eigenvalue of A if and only if it satisfies the equation*

$$\det(A - \lambda I) = 0 \quad (2.9)$$

*This is called the **characteristic equation** of A .*

Example 2.22. Finding Eigenvalues

$$A = \begin{bmatrix} 3 & 0 \\ 8 & -1 \end{bmatrix}.$$

It follows from (2.9) that the eigenvalues of A are the solutions of the equation $A - \lambda I = 0$, which we can write as

$$\begin{vmatrix} \lambda - 3 & 0 \\ -8 & \lambda + 1 \end{vmatrix} = 0$$

from which we obtain

$$(\lambda - 3)(\lambda + 1) = 0$$

This shows that the eigenvalues of A are $\lambda = 3$ and $\lambda = -1$.

Theorem 2.23 ([11]). *Let $\frac{\partial f}{\partial x}(x)$ be the Jacobian matrix for (2.7) evaluated at an equilibrium point \mathbf{x}^* and $\lambda_1, \lambda_2, \dots, \lambda_n$ be its eigenvalues.*

- (i) *If $\operatorname{Re}(\lambda_i) < 0$ for all $i \in \{1, 2, \dots, n\}$, then the equilibrium point \mathbf{x}^* is stable.*
- (ii) *If $\operatorname{Re}(\lambda_i) > 0$ for at least one $i \in \{1, 2, \dots, n\}$, then the equilibrium point \mathbf{x}^* is unstable.*
- (iii) *If $\operatorname{Re}(\lambda_i) = 0$ for at least one $i \in \{1, 2, \dots, n\}$, then the stability of the equilibrium point \mathbf{x}^* cannot be determined by the linearization method.*

Example 2.24. Consider the linear dynamical system

$$\frac{dx}{dt} = Ax,$$

where the matrix A is given by

$$A = \begin{bmatrix} -2 & 1 \\ 0 & -3 \end{bmatrix}.$$

The characteristic equation of A is given by

$$\det(A - \lambda I) = 0.$$

Expanding the determinant,

$$\begin{vmatrix} -2 - \lambda & 1 \\ 0 & -3 - \lambda \end{vmatrix} = (-2 - \lambda)(-3 - \lambda) = 0.$$

Solving for λ , we obtain the eigenvalues:

$$\lambda_1 = -2, \quad \lambda_2 = -3.$$

According to Theorem 2.23, we analyze the real parts of the eigenvalues:

- If $\operatorname{Re}(\lambda_i) < 0$ for all eigenvalues, then the equilibrium point is stable.
- If $\operatorname{Re}(\lambda_i) > 0$ for at least one eigenvalue, then the equilibrium point is unstable.
- If $\operatorname{Re}(\lambda_i) = 0$ for at least one eigenvalue, the stability cannot be determined by linearization.

Since both eigenvalues satisfy $\operatorname{Re}(\lambda_1) = -2 < 0$ and $\operatorname{Re}(\lambda_2) = -3 < 0$, the equilibrium point $x^* = 0$ is stable.

We can confirm the correctness of the solution using numerical methods. Here, we use the fourth-order Runge-Kutta method with step size $h = 0.1$ and initial condition $x(0) = 1$. The computed values are shown in Table 2.1, and the overall trend can be observed in Figure 2.3.

Time (t)	x_1	x_2
0.0	1.000000	1.000000
0.1	0.896629	0.740838
0.2	0.791808	0.548840
0.3	0.691032	0.406601
0.4	0.597444	0.301226

Table 2.1: First five numerical solutions of the system (2.9) computed using the RK4 method.

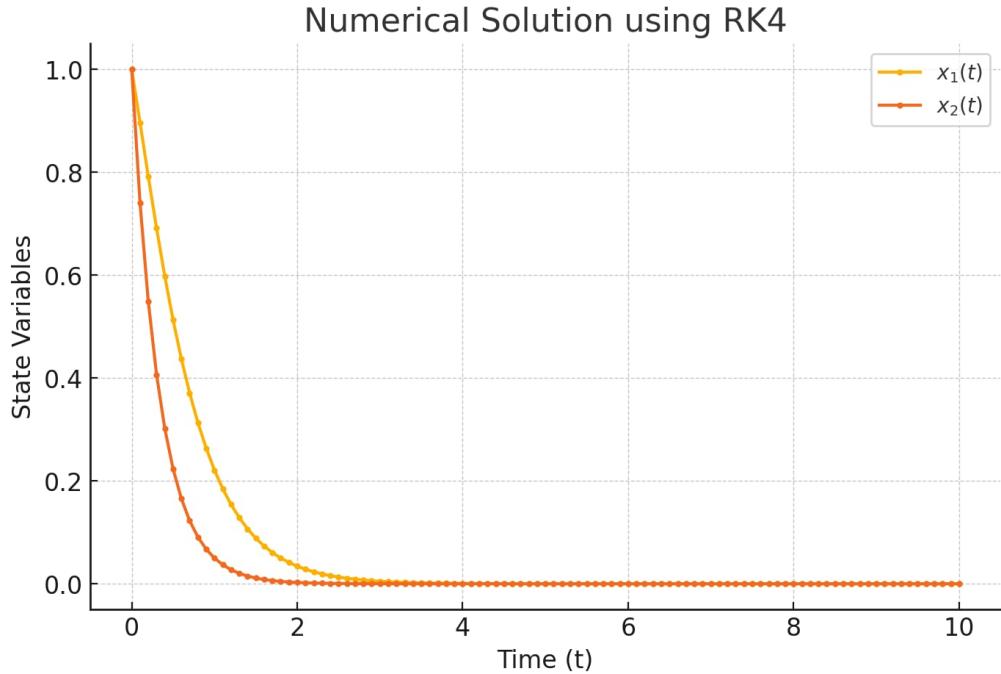


Figure 2.3: Numerical solution of the system (2.9) obtained using the RK4 method.

2.6 Chaotic Dynamical Systems

In the study of dynamical systems, certain systems exhibit chaotic behavior, which is characterized by sensitive dependence on initial conditions, topological mixing, and dense periodic orbits. One of the most well-known chaotic systems is the Lorenz system, introduced by Edward Lorenz in 1963. In this section, we will analyze the chaotic behavior of the Lorenz system to enhance our understanding of chaos.

The Lorenz system is a dynamical system defined by:

$$\begin{aligned}\dot{x}_1 &= \sigma(x_2 - x_1), \\ \dot{x}_2 &= \rho x_1 - x_2 - x_1 x_3, \\ \dot{x}_3 &= x_1 x_2 - \beta x_3.\end{aligned}\tag{2.10}$$

Where the parameters $\sigma > 0$, $\rho > 0$, and $\beta > 0$. The system (2.10) is nonlinear due to the presence of terms like $x_1 x_2$ and $x_1 x_3$, indicating complex interactions among the variables.

Firstly, we determine the equilibrium points of system (2.10) to analyze the chaotic behavior by setting $\dot{x}_1 = 0$, $\dot{x}_2 = 0$, and $\dot{x}_3 = 0$:

$$\sigma(x_2 - x_1) = 0, \quad (2.11)$$

$$\rho x_1 - x_2 - x_1 x_3 = 0, \quad (2.12)$$

$$x_1 x_2 - \beta x_3 = 0. \quad (2.13)$$

From (2.11), we obtain:

$$\sigma(x_2 - x_1) = 0,$$

$$x_2 - x_1 = 0,$$

$$x_2 = x_1.$$

Substituting $x_2 = x_1$ into (2.12), we get:

$$\rho x_1 - x_1 - x_1 x_3 = 0,$$

$$x_1(\rho - 1 - x_3) = 0.$$

This equation implies that $x_1 = 0$ or $x_3 = \rho - 1$. We now consider these two cases separately.

Case 1: If $x_1 = 0$, substituting into (2.13) gives:

$$(0)x_2 - \beta x_3 = 0,$$

$$-\beta x_3 = 0,$$

$$x_3 = 0.$$

Thus, one equilibrium point is $(x_1^*, x_2^*, x_3^*) = (0, 0, 0)$.

Case 2: If $x_3 = \rho - 1$, substituting into (2.13) gives:

$$x_1 x_2 - \beta(\rho - 1) = 0,$$

$$x_1 x_1 = \beta(\rho - 1),$$

$$x_1^2 = \beta(\rho - 1),$$

$$x_1 = \pm \sqrt{\beta(\rho - 1)}.$$

Since $x_1 = x_2$, we obtain the equilibrium points:

$$(x_1^*, x_2^*, x_3^*) = (\sqrt{\beta(\rho - 1)}, \sqrt{\beta(\rho - 1)}, \rho - 1),$$

and

$$(x_1^*, x_2^*, x_3^*) = (-\sqrt{\beta(\rho - 1)}, -\sqrt{\beta(\rho - 1)}, \rho - 1).$$

These equilibrium points will be critical in further analyzing the system's chaotic behavior.

After determining the equilibrium points of the Lorenz system, we now introduce the bifurcation diagram to analyze its transition to chaotic behavior.

A bifurcation diagram is a visual tool used to study how the qualitative behavior of a dynamical system changes as a function of a control parameter. It illustrates the possible long-term values of a system (such as equilibrium points, periodic orbits, or chaotic attractors) as a function of the bifurcation parameter. Bifurcation diagrams are particularly useful for identifying transitions from regular to chaotic dynamics.

To investigate the chaotic behavior of the Lorenz system, we analyze how the system evolves as a specific parameter is varied, leading to bifurcations that eventually result in chaos. The bifurcation diagram is constructed by varying this parameter and plotting the long-term values of one of the system's variables against the parameter. The procedure is as follows:

1. Choose a range of values for the bifurcation parameter.
2. For each value of the parameter, numerically solve the Lorenz system.
3. Record the equilibrium points or long-term values of a selected variable from the numerical solution.
4. Plot these recorded values of the variable against the corresponding parameter values.

Consider the Lorenz system with $\sigma = 10$ and $\beta = \frac{8}{3}$. We vary ρ from 0 to 40 and construct the bifurcation diagram for x_3 , As illustrated in figure 2.4. The resulting diagram illustrates the transition from stable fixed points to periodic orbits and eventually to chaotic behavior as ρ increases. Similarly, we can analyze the parameter σ with $\beta = \frac{8}{3}$ and $\rho = 28$, as well as β with $\sigma = 10$ and $\rho = 28$, using the same procedure. The results are shown in Figures 2.5 and 2.6.

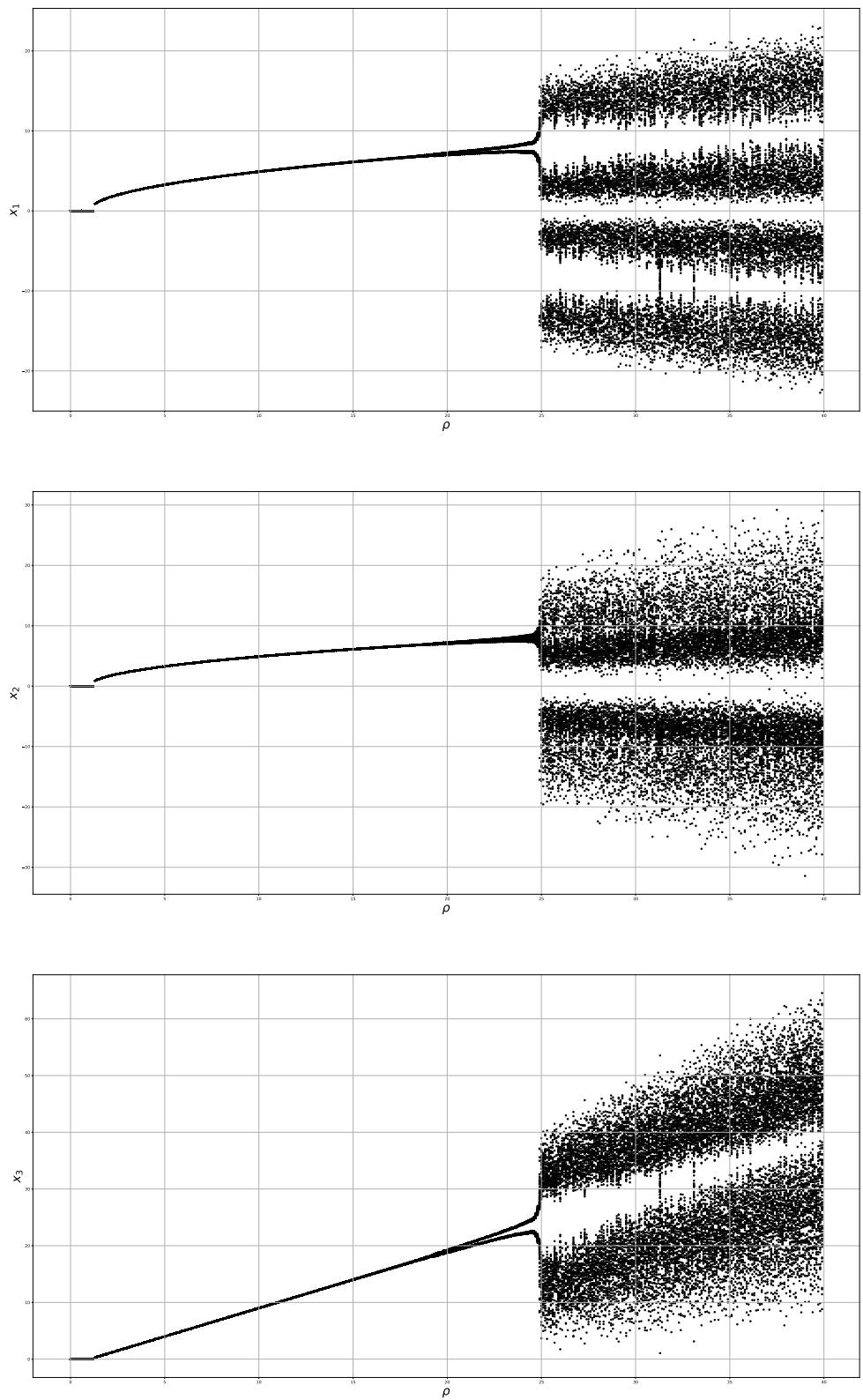


Figure 2.4: Bifurcation diagram of the Lorenz system for ρ ranging from 0 to 40.

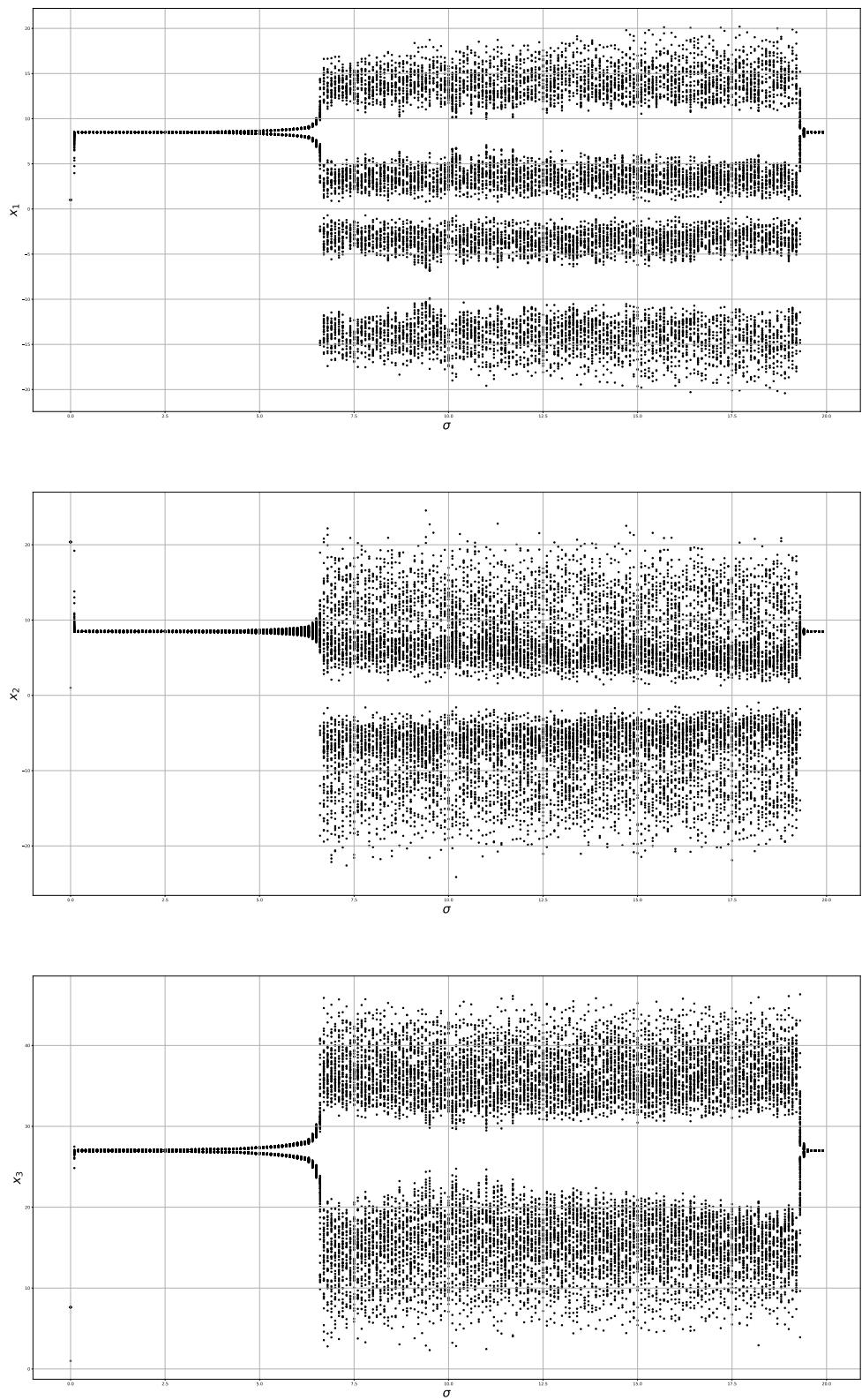


Figure 2.5: Bifurcation diagram of the Lorenz system for σ ranging from 0 to 20.

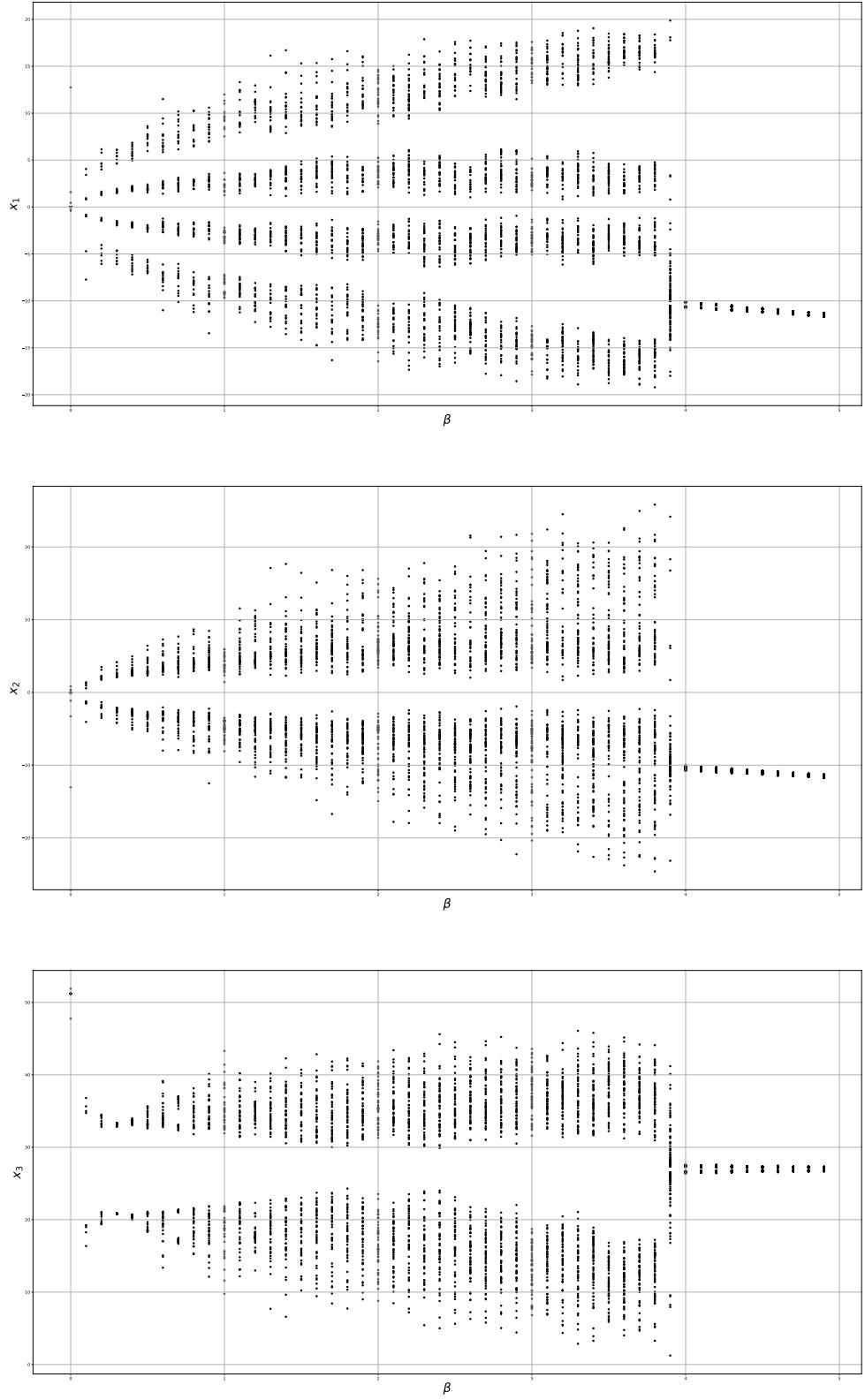


Figure 2.6: Bifurcation diagram of the Lorenz system for β ranging from 0 to 5.

Let us consider an example to clarify why chaotic behavior occurs within certain ranges of the parameter values using Figure 2.4, where the values of x_1 , x_2 and x_3 exhibit multiple points when $\rho \geq 24.95$. We simulate the Lorenz system with the initial condition $(1, 1, 1)$ and parameters $\rho = 22$, $\rho = 23$, $\rho = 24$, and $\rho = 25$. The results are shown in Figure 2.7.

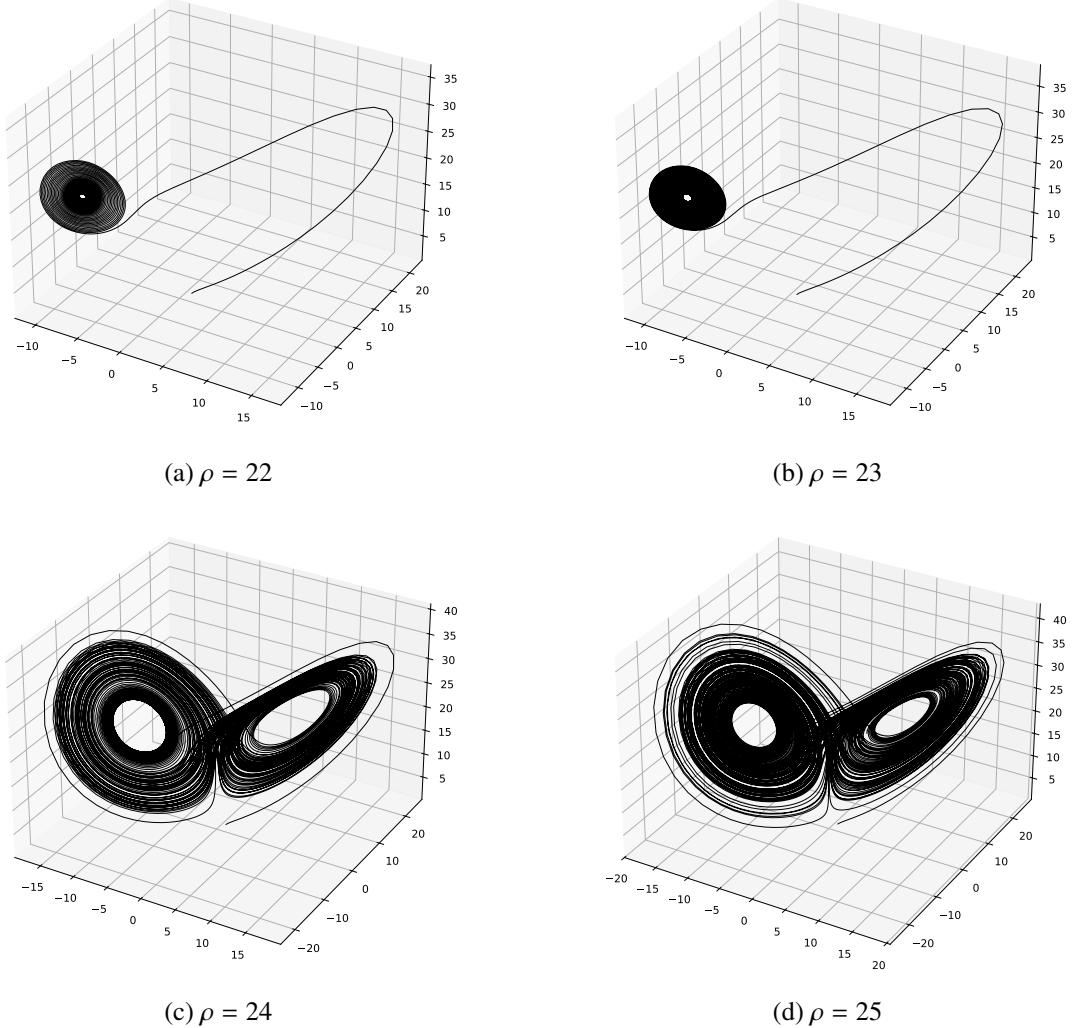


Figure 2.7: The Lorenz system with the initial condition $(1, 1, 1)$ and parameters $\rho = 22$, $\rho = 23$, $\rho = 24$, and $\rho = 25$.

In Figures 2.7a and 2.7b, when the parameter $\rho = 22$ and $\rho = 23$, the Lorenz system shows trajectories that approach a fixed point, as indicated by the circular patterns on the left side of the graphs. However, in Figures 2.7c and 2.7d, when the parameter $\rho = 24$ and $\rho = 25$, the trajectories no longer approach a fixed point. Instead, they form two distinct circular patterns, indicating that the system's behavior becomes unpredictable. These results demonstrate that when $\rho = 24$ and $\rho = 25$, the Lorenz system exhibits chaotic behavior, as confirmed by the bifurcation diagram.

From the bifurcation diagram, a more rigorous analysis can be conducted by examining the eigenvalues of the Jacobian matrix to assess the stability of equilibrium points and the onset of chaotic behavior in the Lorenz system. The Jacobian matrix is derived by linearizing the system of equations (2.10) around its equilibrium points. To facilitate the computation of the Jacobian matrix, the system functions are defined as follows:

$$\begin{aligned} f_1 &= \sigma(x_2 - x_1), \\ f_2 &= \rho x_1 - x_2 - x_1 x_3, \\ f_3 &= x_1 x_2 - \beta x_3. \end{aligned}$$

The general form of the Jacobian matrix for the Lorenz system is then given by:

$$\frac{\partial f}{\partial x}(x_1^*, x_2^*, x_3^*) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x_1^*, x_2^*, x_3^*) & \frac{\partial f_1}{\partial x_2}(x_1^*, x_2^*, x_3^*) & \frac{\partial f_1}{\partial x_3}(x_1^*, x_2^*, x_3^*) \\ \frac{\partial f_2}{\partial x_1}(x_1^*, x_2^*, x_3^*) & \frac{\partial f_2}{\partial x_2}(x_1^*, x_2^*, x_3^*) & \frac{\partial f_2}{\partial x_3}(x_1^*, x_2^*, x_3^*) \\ \frac{\partial f_3}{\partial x_1}(x_1^*, x_2^*, x_3^*) & \frac{\partial f_3}{\partial x_2}(x_1^*, x_2^*, x_3^*) & \frac{\partial f_3}{\partial x_3}(x_1^*, x_2^*, x_3^*) \end{bmatrix} = \begin{bmatrix} -\sigma & \sigma & 0 \\ \rho - x_3^* & -1 & -x_1^* \\ x_2^* & x_1^* & -\beta \end{bmatrix}.$$

where x_1^*, x_2^*, x_3^* is an equilibrium point of the Lorenz system. To analyze the stability of the Lorenz system in the regime where chaotic behavior emerges, we consider the parameter values $\sigma = 10$, $\rho = 28$, and $\beta = \frac{8}{3}$, which are well known to induce chaotic dynamics. The Jacobian matrix of the system is given by:

$$\frac{\partial f}{\partial x}(x_1^*, x_2^*, x_3^*) = \begin{bmatrix} -10 & 10 & 0 \\ 28 - x_3^* & -1 & -x_1^* \\ x_2^* & x_1^* & -\frac{8}{3} \end{bmatrix}. \quad (2.14)$$

The eigenvalues are computed by solving the characteristic equation $\det(\frac{\partial f}{\partial x}(x_1^*, x_2^*, x_3^*) - \lambda I) = 0$. For convenience, we use the eigenvalues at the equilibrium points from [12] as follows:

1. The eigenvalues at the equilibrium point $(0, 0, 0)$ are:

$$\lambda_1 = 11.8277,$$

$$\lambda_2 = -22.8277,$$

$$\lambda_3 = -2.6667.$$

2. The equilibrium points $(\sqrt{\beta(\rho - 1)}, \sqrt{\beta(\rho - 1)}, \rho - 1)$ and $(-\sqrt{\beta(\rho - 1)}, -\sqrt{\beta(\rho - 1)}, \rho - 1)$ with $\sigma = 10$, $\rho = 28$, and $\beta = \frac{8}{3}$, the eigenvalues are:

$$\lambda_1 = -13.8545,$$

$$\lambda_2 = 0.0939 + 10.1945i,$$

$$\lambda_3 = 0.0939 - 10.1945i.$$

where i is the imaginary unit.

These eigenvalues help determine whether the Lorenz system exhibits chaotic behavior. Specifically, the conditions for chaotic behavior are:

1. At the equilibrium point $(0, 0, 0)$, there exists at least one positive real eigenvalue and at least one negative real eigenvalue.
2. At the equilibrium points $(\pm \sqrt{\beta(\rho - 1)}, \pm \sqrt{\beta(\rho - 1)}, \rho - 1)$, there exists a pair of complex conjugate eigenvalues with positive real parts and one negative real eigenvalue.

2.7 Music Theory

Music theory provides the foundation for understanding the organization and notation of musical sounds. One of the most fundamental concepts in music is the musical note, which represents both pitch (how high or low a sound is) and duration (how long the sound lasts).

2.7.1 Pitch

Pitch refers to the perceived frequency of a sound, represented symbolically in musical notation. In Western music, pitches are named using the letters A through G. The standard sequence of pitches is as follows:

$$C, D, E, F, G, A, B.$$

This sequence is ordered from the lowest to the highest sound. The comparison of pitches can be denoted using the symbol $x < y$, which indicates that the pitch x is lower than the pitch y . For example:

Example 2.25. In music theory, the following comparison holds:

$$C < D < E < F < G < A < B.$$

This demonstrates that C is the lowest pitch, and B is the highest pitch in this sequence.

2.7.2 Octaves

An octave refers to a group of twelve notes that repeat at different pitch levels. The standard octave is represented as:

$$C, C\sharp, D, D\sharp, E, F, F\sharp, G, G\sharp, A, A\sharp, B,$$

or equivalently:

$$C, D\flat, E\flat, F, G\flat, G, A\flat, A, B\flat, B.$$

Each octave is numbered, starting from octave 0 (very low pitch) and increasing to octave 10 (very high pitch). Formally, an octave O_n is defined as:

$$O_n = \{Cn, C\sharp n, Dn, D\sharp n, En, Fn, F\sharp n, Gn, G\sharp n, An, A\sharp n, Bn\},$$

or equivalently,

$$O_n = \{Cn, D\flat n, Dn, E\flat n, En, Fn, G\flat n, Gn, A\flat n, An, B\flat n, Bn\}.$$

The equivalence between enharmonic notes within the octave structure is expressed by the following relationships:

$$\begin{aligned} C\sharp n &= D\flat n, \\ D\sharp n &= E\flat n, \\ F\sharp n &= G\flat n, \\ G\sharp n &= A\flat n, \\ A\sharp n &= B\flat n. \end{aligned}$$

These enharmonic equivalences illustrate the dual notation commonly used in Western music theory, where certain pitch classes can be represented by multiple names depending on their harmonic and tonal context.

Example 2.26. The first octave O_1 is expressed as:

$$C1, C\sharp 1, D1, D\sharp 1, E1, F1, F\sharp 1, G1, G\sharp 1, A1, A\sharp 1, B1.$$

Which the following comparison holds:

$$C1 < C\sharp 1 < D1 < D\sharp 1 < E1 < F1 < F\sharp 1 < G1 < G\sharp 1 < A1 < A\sharp 1 < B1.$$

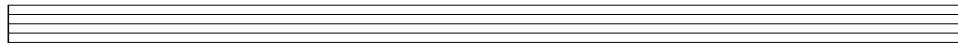
Example 2.27. The octaves are compared as follows:

$$O_0 < O_1 < O_2 < O_3 < O_4 < O_5 < O_6 < O_7 < O_8 < O_9 < O_{10},$$

which indicates that O_0 represents the lowest sound and O_{10} represents the highest.

2.7.3 The Staff and Clefs

The staff is a five horizontal lines used to notate musical notes and their durations. The lines and spaces represent different pitches. The standard five-line staff is shown below:



The clef symbol placed at the beginning of the staff indicates the pitch range of the notes. The two primary clefs are:

- Treble Clef ($\text{\textcircled{G}}$): Used for higher-pitched notes (e.g., violin, flute).
- Bass Clef ($\text{\textcircled{F}}$): Used for lower-pitched notes (e.g., bass guitar, tuba).

The following example illustrates how notes are read on the staff for both clefs:

Example 2.28. The upper staff employs the Treble Clef ($\text{\textcircled{G}}$), whereas the lower staff utilizes the Bass Clef ($\text{\textcircled{F}}$). In this example, let \bullet represent a musical note. The pitch name corresponding to each line and space on the staff is illustrated below:

Moreover, the interpretation of a musical note is contingent upon the clef in use. Additionally, the standard five-line staff may incorporate ledger lines above or below the main staff to extend the range of notated pitches.

2.7.4 Duration

Duration refers to the length of time a pitch is played. In music theory, duration is typically measured in beats. Let us define one beat as one second for convenience. The standard note durations are as follows:

- The **whole note** (\circ) is a musical symbol that represents a duration of 4 beats, equivalent to 4 seconds.
- The **half note** ($\text{\textcircled{D}}$) is a notation that indicates a playing time of 2 beats, which corresponds to 2 seconds.
- The **quarter note** (\bullet) denotes a duration of 1 beat, lasting for 1 second.
- The **eighth note** ($\text{\textcircled{J}}$) signifies a duration of $\frac{1}{2}$ of a beat, or $\frac{1}{2}$ of a second.
- The **sixteenth note** ($\text{\textcircled{N}}$) represents a duration of $\frac{1}{4}$ of a beat, corresponding to $\frac{1}{4}$ of a second.
- The **thirty-second note** ($\text{\textcircled{M}}$) is a musical notation that indicates a duration of $\frac{1}{8}$ of a beat, equal to $\frac{1}{8}$ of a second.

- The **sixty-fourth note** (♪) is a note that represents a duration of $\frac{1}{16}$ of a beat, lasting for $\frac{1}{16}$ of a second.

The comparison of note durations is as follows:

Example 2.29. Consider the following sequence of musical notes notated on a five-line staff using the Treble Clef (G):



In standard notation, musical notes are read from left to right. The sequence begins with the whole note (o), which has a duration of 4 seconds. Upon completion of the whole note, the half note (J) follows, indicating a duration of 2 seconds. This pattern continues with the quarter note (J) lasting for 1 second, the eighth note (J) for $\frac{1}{2}$ second, the sixteenth note (J) for $\frac{1}{4}$ second, the thirty-second note (J) for $\frac{1}{8}$ second, and finally, the sixty-fourth note (♪) for $\frac{1}{16}$ second.

From this sequence, it is evident that the ♪ has the shortest duration, whereas the o has the longest duration.

Rests

Rests represent periods of silence in music, with durations analogous to those of notes. The standard rests are as follows:

- The **whole rest** (-) is a musical symbol that indicates a silence duration of 4 beats, equivalent to 4 seconds.
- The **half rest** (-) represents a silence lasting for 2 beats, corresponding to 2 seconds.
- The **quarter rest** (:) denotes a silence duration of 1 beat, equivalent to 1 second.
- The **eighth rest** (;) signifies a period of silence lasting for $\frac{1}{2}$ of a beat, or $\frac{1}{2}$ of a second.
- The **sixteenth rest** (;) represents a silence duration of $\frac{1}{4}$ of a beat, corresponding to $\frac{1}{4}$ of a second.
- The **thirty-second rest** (;) is a musical notation that indicates a silence of $\frac{1}{8}$ of a beat, equal to $\frac{1}{8}$ of a second.
- The **sixty-fourth rest** (;) is a rest that represents a silence duration of $\frac{1}{16}$ of a beat, lasting for $\frac{1}{16}$ of a second.

Rests are compared similarly to notes:

Example 2.30. Consider the following sequence of musical notes notated on a five-line staff using the Treble Clef (G):



In standard notation, musical notes are read from left to right. The sequence begins with the whole rest (-), which has a silence duration of 4 seconds. Upon completion of the whole rest, the half rest (-) follows, indicating a silence duration of 2 seconds. This pattern continues with the quarter rest (q) lasting for 1 second, the eighth rest (r) for $\frac{1}{2}$ second, the sixteenth rest (s) for $\frac{1}{4}$ second, the thirty-second note (t) for $\frac{1}{8}$ second, and finally, the sixty-fourth note (u) for $\frac{1}{16}$ second.

From this sequence, it is evident that the u has the shortest duration, whereas the - has the longest duration.

Dotted Notes

A dotted note increases the duration of the original note by half of its value. The standard durations for dotted notes are as follows:

- The **dotted whole note** (o.) represents a duration of 6 beats, equivalent to 6 seconds.
- The **dotted half note** (d.) extends the half note's duration to 3 beats, corresponding to 3 seconds.
- The **dotted quarter note** (d) indicates a duration of 1.5 beats, lasting for 1.5 seconds.
- The **dotted eighth note** (d.) signifies a duration of $\frac{3}{4}$ of a beat, or $\frac{3}{4}$ of a second.
- The **dotted sixteenth note** (d.) represents a duration of $\frac{3}{8}$ of a beat, corresponding to $\frac{3}{8}$ of a second.
- The **dotted thirty-second note** (d.) is a musical notation that indicates a duration of $\frac{3}{16}$ of a beat, equal to $\frac{3}{16}$ of a second.
- The **dotted sixty-fourth note** (d.) represents a duration of $\frac{3}{32}$ of a beat, lasting for $\frac{3}{32}$ of a second.

The comparison of dotted notes is as follows:

Example 2.31. Consider the following sequence of musical notes notated on a five-line staff using the Treble Clef (G):



In standard notation, musical notes are read from left to right. The sequence begins with the dotted whole note (o.), which has a duration of 6 seconds. Upon completion of the dotted whole note, the dotted half note (d.) follows, indicating a duration of 3 seconds. This pattern continues with the dotted quarter note (d) lasting for 1.5 second, the dotted eighth note (d.) for $\frac{3}{4}$ second,

the dotted sixteenth note (♪) for $\frac{3}{8}$ second, the dotted thirty-second note (♪) for $\frac{3}{16}$ second, and finally, the dotted sixty-fourth note (♪) for $\frac{3}{32}$ second.

From this sequence, it is evident that the ♪ has the shortest duration, whereas the ♩ has the longest duration.

2.8 Musical Variations

Variation is a formal technique in which musical material is repeated in an altered form. In music composition, variation serves as a fundamental method for transforming an original theme while maintaining a sense of coherence. The alterations can range from subtle modifications to significant structural changes, allowing composers to explore different expressive possibilities while retaining a recognizable link to the initial material. This technique has been widely used across various musical genres and historical periods, from classical compositions to modern experimental music.

Changes in variation may affect multiple musical elements, including melody, rhythm, harmony, counterpoint, timbre, and orchestration. A variation can involve modifying a single musical component or multiple elements simultaneously. For example, a composer might alter the melody by adding embellishments or shifting pitches while keeping the harmonic foundation intact. In contrast, orchestration changes may involve reassigning a theme to different instruments, thereby altering its texture and timbral quality. The diverse ways in which variations can be implemented provide musicians with endless creative possibilities.

There are various techniques for creating musical variations, each producing distinct effects. Melodic variation, for instance, involves changing the contour or intervals of the original melody while preserving its fundamental structure. Rhythmic variation, on the other hand, introduces modifications to note durations, syncopation, or rhythmic subdivisions, sometimes adding extra notes or altering metric accents. These techniques can be combined or applied independently to create unique interpretations of a musical theme.

Melodic and rhythmic variations play essential roles in musical development and artistic expression. Composers often use these techniques to introduce contrast, maintain listener interest, and develop thematic material throughout a piece. By applying variation techniques, musicians can generate new perspectives on familiar melodies, resulting in fresh and dynamic performances. Figure 2.8 illustrates examples of variations, showcasing how these transformations contribute to the richness of musical composition.



(a) The original of Ah vous dirai-je Maman in the first 8 bars.



(b) Ah vous dirai-je Maman with melodic variation in the first 8 bars.



(c) Ah vous dirai-je Maman with rhythmic variation in the first 8 bars.

Susensions

A musical score for piano in common time (2/4). The key signature changes to one flat (B-flat major). The top staff shows a melodic line with 'Suspensions' and 'Imitation'. The bottom staff shows harmonic support with 'Independent melodic line (counterpoint)'. The piano part includes bass notes and chords.

(d) Ah vous dirai-je Maman with Minor mode variation in the first 8 bars.

Figure 2.8: The original and variation example.

Chapter 3

Main Results

This chapter outlines the procedure for generating musical variations from a chaotic mapping and the incorporation of musical variation with expanded rhythm. The first technique demonstrates how a chaotic mapping can be utilized to create new variations in musical pitch. Subsequently, the second technique combines the procedure of generating musical variations from a chaotic mapping with musical variation involving expanded rhythm, thereby producing more intricate and captivating musical variations.

3.1 Musical Variations from a Chaotic Mapping Method

This section presents an adaptation of the method proposed by Dabby [13] for generating musical variations by exploiting chaotic dynamics. Consider a sheet music with m being a positive integer representing the number of notes, and $\{p_k\}_{k=0}^{m-1}$ being a sequence of musical pitches. Let

$$\dot{x}(t) = f(t, x) \quad (3.1)$$

define a chaotic dynamical system with an initial condition $x(0) = x_0 \in \mathbb{R}^n$, where $f : \mathbb{R}_+ \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a continuous function.

Given a sequence $\{\phi_i(kh)\}_{k=0}^{m-1}$, where $\phi_i : h\mathbb{N}_{m-1} \rightarrow \mathbb{R}$ is a numerical solution in the i -th component of the system (3.1) for some $i \in \mathbb{N}_n$ and a step size h , we define a mapping α as

$$\alpha(\phi_i(kh)) := p_k \quad (3.2)$$

for all $k \in \{0\} \cup \mathbb{N}_{m-1}$.

To generate a new variation of the music, we consider $\{\tilde{\phi}_i(kh)\}_{k=0}^{m-1}$ as a sequence of a new trajectory with the initial condition $x(0) = \tilde{x}_0 \in \mathbb{R}^n$, where $\tilde{\phi}_i$ is a numerical solution in the i -th component to the system (3.1), and \tilde{x}_0 is located not far from x_0 , i.e., $\|x_0 - \tilde{x}_0\| \leq d$ for some small positive number $d \in \mathbb{R}$. We then define another mapping β as:

$$\beta(\tilde{\phi}_i(kh)) := \begin{cases} \alpha(\phi_i(b)) & \text{if } \exists a, b \in \text{dom}(\phi_i; m, h) \text{ s.t. } \phi_i(a) < \tilde{\phi}_i(kh) \leq \phi_i(b) \\ & \text{and } \nexists c \in \text{dom}(\phi_i; m, h) \text{ s.t. } \phi_i(a) < \phi_i(c) \leq \phi_i(b), \\ \alpha(\min\{\phi_i(t)\}) & \text{if } \tilde{\phi}_i(kh) < \phi_i(t) \text{ for all } t \in \text{dom}(\phi_i; m, h), \\ \alpha(\max\{\phi_i(t)\}) & \text{if } \phi_i(t) < \tilde{\phi}_i(kh) \text{ for all } t \in \text{dom}(\phi_i; m, h), \end{cases} \quad (3.3)$$

where $\text{dom}(\phi_i; m, h) = \{t \in \text{dom } \phi_i : t \leq (m-1)h\}$. Resulting in the sequence $\{\beta(\tilde{\phi}_i(kh))\}_{k=0}^{m-1}$, which represents a new variation of the original musical pitch.

Example 3.1. Consider the first three bars of the sheet music for the music “Ah vous dirai-je Maman” [14], illustrated in Figure 3.3a. The sequence

$$\{p_k\}_{k=0}^{10} = \{C4, C4, G4, G4, A4, A4, G4, F4, F4, E4, E4\}$$

represents the pitches in this excerpt. Let

$$\dot{x}(t) = f(t, x) = \begin{bmatrix} f_1(t, x) \\ f_2(t, x) \\ f_3(t, x) \end{bmatrix} := \begin{bmatrix} 10(x_2 - x_1) \\ 28x_1 - x_2 - x_1x_3 \\ x_1x_2 - 2.6667x_3 \end{bmatrix} \quad (3.4)$$

where $x = (x_1, x_2, x_3) \in \mathbb{R}^3$. Firstly, we consider the numerical solution in the first component to the system (3.4) $\phi_1(t)$, derived using the fourth-order Runge-Kutta method with an initial condition $x(0) = (1, 1, 1)$ and a step size $h = 0.01$. The first eleven values of ϕ_1 are represented by the sequence

$$\begin{aligned}\{\phi_1(kh)\}_{k=0}^{10} &= \{\phi_1(0), \phi_1(0.01), \phi_1(0.02), \phi_1(0.03), \phi_1(0.04), \phi_1(0.05), \phi_1(0.06), \phi_1(0.07), \\ &\quad \phi_1(0.08), \phi_1(0.09), \phi_1(0.1)\} \\ &= \{1.00, 1.29, 2.13, 3.74, 6.54, 11.04, 16.69, 19.56, 15.37, 7.55, 1.20\}.\end{aligned}$$

We therefore define a mapping α as follows:

$$\alpha(\phi_1(kh)) := p_k.$$

For instance, the first element in $\{\phi_1(kh)\}_{k=0}^{10}$ is mapped to the first element in $\{p_k\}_{k=0}^{10}$, the second element in $\{\phi_1(kh)\}_{k=0}^{10}$ is mapped to the second element in $\{p_k\}_{k=0}^{10}$. Following this mapping, we obtain:

$$\begin{aligned}\alpha(\phi_1(0)) &= \alpha(1.00) = C4, \\ \alpha(\phi_1(0.01)) &= \alpha(1.29) = C4, \\ \alpha(\phi_1(0.02)) &= \alpha(2.13) = G4, \\ \alpha(\phi_1(0.03)) &= \alpha(3.74) = G4, \\ \alpha(\phi_1(0.04)) &= \alpha(6.54) = A4, \\ \alpha(\phi_1(0.05)) &= \alpha(11.04) = A4, \\ \alpha(\phi_1(0.06)) &= \alpha(16.69) = G4, \\ \alpha(\phi_1(0.07)) &= \alpha(19.56) = F4, \\ \alpha(\phi_1(0.08)) &= \alpha(15.37) = F4, \\ \alpha(\phi_1(0.09)) &= \alpha(7.55) = E4, \\ \alpha(\phi_1(0.10)) &= \alpha(1.20) = E4.\end{aligned}$$

A summary of these mappings is provided in Table 3.1, while the overall mapping α is illustrated in Figure 3.1.

Next, we consider the numerical solution in the first component to the system (3.4) $\tilde{\phi}_1(t)$, derived again using the fourth-order Runge-Kutta method with the same step size h , but with a different initial condition $x(0) = (1.01, 1, 1)$. The first eleven values of $\tilde{\phi}_1$ is represented as

$$\begin{aligned}\{\tilde{\phi}_1(kh)\}_{k=0}^{10} &= \{\tilde{\phi}_1(0), \tilde{\phi}_1(0.01), \tilde{\phi}_1(0.02), \tilde{\phi}_1(0.03), \tilde{\phi}_1(0.04), \tilde{\phi}_1(0.05), \tilde{\phi}_1(0.06), \tilde{\phi}_1(0.07), \\ &\quad \tilde{\phi}_1(0.08), \tilde{\phi}_1(0.09), \tilde{\phi}_1(0.1)\} \\ &= \{1.01, 1.30, 2.15, 3.76, 6.58, 11.10, 16.73, 19.55, 15.30, 7.48, 1.15\}.\end{aligned}$$

Let β be a mapping defined as (3.3), we note that

$$\in \text{dom}(\phi_1; 11, 0.01) = \{0, 0.01, 0.02, \dots, 0.1\}.$$

To illustrate this mapping, consider the following cases:

1. For $\tilde{\phi}_1(0) = 1.01$, we observe that there exist $0, 0.1 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0) < \tilde{\phi}_1(0) \leq \phi_1(0.1),$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0) < \phi_1(c) \leq \phi_1(0.1).$$

This yield:

$$\beta(\tilde{\phi}_1(0)) = \beta(1.01) = \alpha(\phi_1(0.1)) = \alpha(1.20) = E4.$$

2. For $\tilde{\phi}_1(0.01) = 1.30$, we observe that there exist $0.01, 0.02 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0.01) < \tilde{\phi}_1(0.01) \leq \phi_1(0.02),$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0.01) < \phi_1(c) \leq \phi_1(0.02).$$

This yield:

$$\beta(\tilde{\phi}_1(0.01)) = \beta(1.30) = \alpha(\phi_1(0.02)) = \alpha(2.13) = G4.$$

3. For $\tilde{\phi}_1(0.02) = 2.15$, we observe that there exist $0.02, 0.03 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0.02) < \tilde{\phi}_1(0.02) \leq \phi_1(0.03) = 3.74,$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0.02) < \phi_1(c) \leq \phi_1(0.03).$$

This yield:

$$\beta(\tilde{\phi}_1(0.02)) = \beta(2.15) = \alpha(\phi_1(0.03)) = \alpha(3.74) = G4.$$

4. For $\tilde{\phi}_1(0.03) = 3.76$, we observe that there exist $0.03, 0.04 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0.03) < \tilde{\phi}_1(0.03) \leq \phi_1(0.04) = 6.54,$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0.03) < \phi_1(c) \leq \phi_1(0.04).$$

This yield:

$$\beta(\tilde{\phi}_1(0.03)) = \beta(3.76) = \alpha(\phi_1(0.04)) = \alpha(6.54) = A4.$$

5. For $\tilde{\phi}_1(0.04) = 6.58$, we observe that there exist $0.04, 0.09 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0.04) < \tilde{\phi}_1(0.04) \leq \phi_1(0.09),$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0.04) < \phi_1(c) \leq \phi_1(0.09).$$

This yield:

$$\beta(\tilde{\phi}_1(0.04)) = \beta(6.58) = \alpha(\phi_1(0.09)) = \alpha(7.48) = E4.$$

6. For $\tilde{\phi}_1(0.05) = 11.10$, we observe that there exist $0.05, 0.08 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0.05) < \tilde{\phi}_1(0.05) \leq \phi_1(0.08),$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0.05) < \phi_1(c) \leq \phi_1(0.08).$$

This yield:

$$\beta(\tilde{\phi}_1(0.05)) = \beta(11.10) = \alpha(\phi_1(0.08)) = \alpha(15.37) = F4.$$

7. For $\tilde{\phi}_1(0.06) = 16.73$, we observe that there exist $0.08, 0.06 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0.08) < \tilde{\phi}_1(0.06) \leq \phi_1(0.06),$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0.08) < \phi_1(c) \leq \phi_1(0.06).$$

This yield:

$$\beta(\tilde{\phi}_1(0.06)) = \beta(16.73) = \alpha(\phi_1(0.06)) = \alpha(16.69) = F4.$$

8. For $\tilde{\phi}_1(0.07) = 19.55$, we observe that there exist $0.06, 0.07 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0.06) < \tilde{\phi}_1(0.07) \leq \phi_1(0.07),$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0.06) < \phi_1(c) \leq \phi_1(0.07).$$

This yield:

$$\beta(\tilde{\phi}_1(0.07)) = \beta(19.55) = \alpha(\phi_1(0.07)) = \alpha(19.56) = F4.$$

9. For $\tilde{\phi}_1(0.08) = 15.30$, we observe that there exist $0.05, 0.08 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0.05) < \tilde{\phi}_1(0.08) \leq \phi_1(0.08),$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0.05) < \phi_1(c) \leq \phi_1(0.08).$$

This yield:

$$\beta(\tilde{\phi}_1(0.08)) = \beta(15.30) = \alpha(\phi_1(0.08)) = \alpha(15.37) = F4.$$

10. For $\tilde{\phi}_1(0.09) = 7.48$, we observe that there exist $0.04, 0.09 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0.04) < \tilde{\phi}_1(0.09) \leq \phi_1(0.09),$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0.04) < \phi_1(c) \leq \phi_1(0.09).$$

This yield:

$$\beta(\tilde{\phi}_1(0.09)) = \beta(7.48) = \alpha(\phi_1(0.09)) = \alpha(7.55) = F4.$$

11. For $\tilde{\phi}_1(0.1) = 1.15$, we observe that there exist $0, 0.1 \in \text{dom}(\phi_1; 11, 0.01)$ such that:

$$\phi_1(0) < \tilde{\phi}_1(0.1) \leq \phi_1(0.1),$$

and since there is no $c \in \text{dom}(\phi_1; 11, 0.01)$ such that

$$\phi_1(0) < \phi_1(c) \leq \phi_1(0.1).$$

This yield:

$$\beta(\tilde{\phi}_1(0.10)) = \beta(1.15) = \alpha(\phi_1(0.1)) = \alpha(1.20) = E4.$$

The complete mapping is summarized as follows:

$$\begin{aligned}
\beta(\tilde{\phi}_1(0)) &= \beta(1.01) = E4, \\
\beta(\tilde{\phi}_1(0.01)) &= \beta(1.30) = G4, \\
\beta(\tilde{\phi}_1(0.02)) &= \beta(2.15) = G4, \\
\beta(\tilde{\phi}_1(0.03)) &= \beta(3.76) = A4, \\
\beta(\tilde{\phi}_1(0.04)) &= \beta(6.58) = E4, \\
\beta(\tilde{\phi}_1(0.05)) &= \beta(11.10) = F4, \\
\beta(\tilde{\phi}_1(0.06)) &= \beta(16.73) = F4, \\
\beta(\tilde{\phi}_1(0.07)) &= \beta(19.55) = F4, \\
\beta(\tilde{\phi}_1(0.08)) &= \beta(15.30) = F4, \\
\beta(\tilde{\phi}_1(0.09)) &= \beta(7.48) = F4, \\
\beta(\tilde{\phi}_1(0.10)) &= \beta(1.15) = E4.
\end{aligned}$$

Additionally, a summary of these mappings is presented in Table 3.1, and the overall mapping process of β is illustrated in Figure 3.2. Consequently, the mapping β produces the sequence:

$$\{\beta(\tilde{\phi}_1(kh))\}_{k=0}^{10} = \{E4, G4, G4, A4, E4, F4, F4, F4, F4, E4, E4\}.$$

This sequence is illustrated in Figure 3.3b.

The visualization of this method is illustrated in Figure 3.4. If we consider the musical notes that change at indices $k \in \{1, 2, 4, 5, 6, 7\}$, the change in pitch compared to the original musical pitches is approximately 54%. However, the audience might not distinguish the difference between the original and new notes as much as the change suggests, which makes this method less suitable for generating more interesting musical variations.

k	0	1	2	3	4	5	6	7	8	9	10
kh	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
$\phi_1(kh)$	1.00	1.29	2.13	3.74	6.54	11.04	16.69	19.56	15.37	7.55	1.20
$\alpha(\phi_1(kh))$	C4	C4	G4	G4	A4	A4	G4	F4	F4	E4	E4
$\tilde{\phi}_1(kh)$	1.01	1.30	2.15	3.76	6.58	11.10	16.73	19.55	15.30	7.48	1.15
$\beta(\tilde{\phi}_1(kh))$	E4	G4	G4	A4	E4	F4	F4	F4	F4	E4	E4

Table 3.1: The result of a mapping g from $\{\phi_1(kh)\}_{k=0}^{10}$ to $\{p_k\}_{k=0}^{10}$ and the result of a mapping l from $\{\tilde{\phi}_1(kh)\}_{k=0}^{10}$ to new musical pitches.

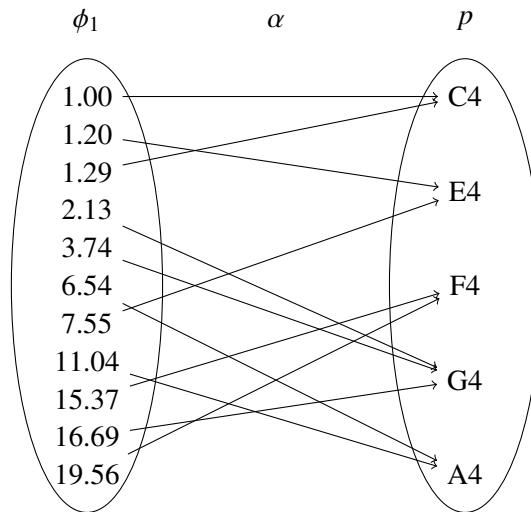


Figure 3.1: Mapping α .

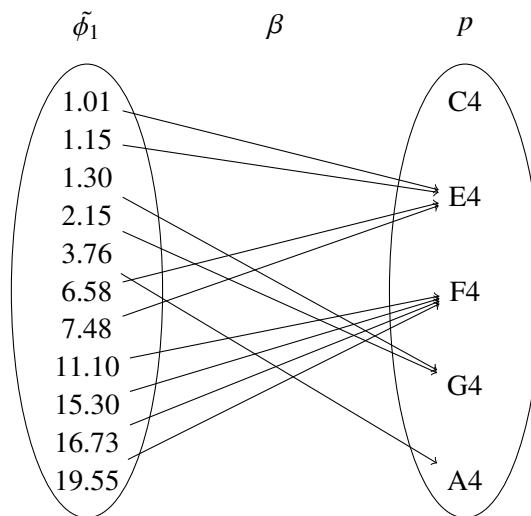


Figure 3.2: Mapping β .

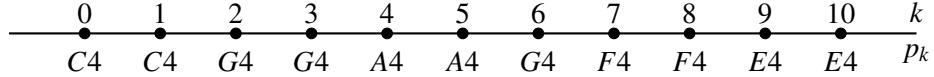


(a) The original of Ah vous dirai-je Maman in the first 3 bars.

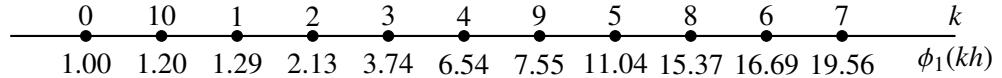


(b) The new variation of Ah vous dirai-je Maman in the first 3 bars, generated by musical variations from a chaotic mapping method with the initial condition (1.01, 1, 1).

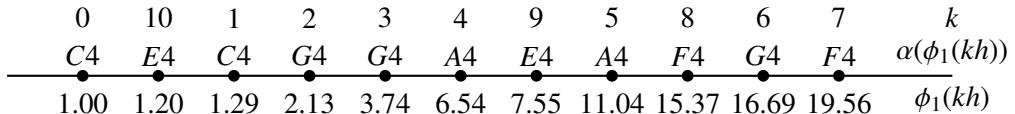
Figure 3.3: The original and new variation of Ah vous dirai-je Maman.



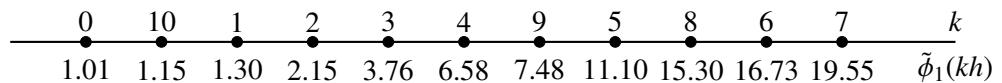
(a) The first 11 pitches of the Ah vous dirai-je Maman are marked below the 1D axis.



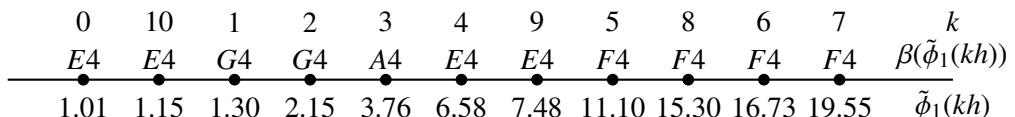
(b) The first 11 numerical solution in first component to the system (3.4) with initial condition of $(1, 1, 1)$ are marked below the 1D axis.



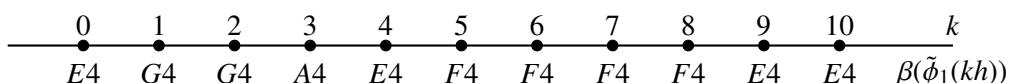
(c) For each component in $\{\phi_1(kh)\}_{k=0}^{10}$, apply the $\alpha(\phi_1(kh))$ mapping.



(d) The first 11 numerical solution in first component to the system (3.4) with new initial condition of $(1.01, 1, 1)$ are marked below the 1D axis.



(e) For each component in $\{\tilde{\phi}_1(kh)\}_{k=0}^{10}$, apply the $\beta(\tilde{\phi}_1(kh))$ mapping.



(f) The new variation of the first 11 pitches are marked below the 1D axis.

Figure 3.4: The figure for visualizes how a chaotic mapping method can be used to generate musical variations.

In the next section, we present a technique for extending the rhythmic duration of musical notes, thereby opening up a number of possibilities for creating new musical variations.

3.2 Expanded Rhythm Method

Let $\{p_k\}_{k=0}^{m-1}$ denote a sequence of music pitches. We can then establish a sequence of expanded music pitches $\{p'_k\}_{k=0}^{mq-1}$, where q is a positive integer representing the number of expanded notes for each p_k . Additionally, we set $p_k = p'_{kq} = p'_{kq+1} = \dots = p'_{q(k+1)-1}$.

Example 3.2. The sequence $\{p_k\}_{k=0}^3 = \{C4, C4, G4, G4\}$ represents the sequence of musical pitches in the first bar of the music piece “Ah vous dirai-je Maman,” as illustrated in Figure 3.5a. Assuming $q = 4$ denotes the number of expanded notes for each p_k , we can establish an expanded sequence of musical pitches

$$\{p'_k\}_{k=0}^{15} = \{C4, C4, C4, C4, C4, C4, C4, C4, G4, G4, G4, G4, G4, G4, G4, G4\},$$

illustrated in Figure 3.5b.



(a) The original of Ah vous dirai-je Maman in the first bars.



(b) The musical variation of Ah vous dirai-je, maman in the first bars.

Figure 3.5: The original and musical variation of Ah vous dirai-je Maman.

3.3 Musical Variations from a Expanded Rhythm Method

This section presents an adaptation of the method proposed in section 3.1 for generating musical variations by expanded rhythm of music note.

Consider a sheet of music where m is a positive integer representing the number of notes and $\{p_k\}_{k=0}^{m-1}$ represents a sequence of musical pitches. Denote the positive integer q where $q > 1$ be expanded notes for each p_k and $\{p'_k\}_{k=0}^{mq-1}$ represents the sequence of expanded rhythms corresponding to the musical pitches. Let

$$\dot{x}(t) = f(t, x) \tag{3.5}$$

define a chaotic dynamical system with an initial condition $x(0) = x_0 \in \mathbb{R}^n$, where $f : \mathbb{R}_+ \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a continuous function.

Given a sequence $\{\phi_i(kh)\}_{k=0}^{mq-1}$, where $\phi_i : h\mathbb{N}_{mq-1} \rightarrow \mathbb{R}$ is a numerical solution in the i -th component of the system (3.5) for some $i \in \mathbb{N}_n$ and a step size h , we define a mapping α as

$$\alpha(\phi_i(kh)) := p'_k \tag{3.6}$$

for all $k \in \{0\} \cup \mathbb{N}_{mq-1}$.

To generate a new variation of the music, we consider $\{\tilde{\phi}_i(kh)\}_{k=0}^{mq-1}$ as a sequence of a new trajectory with the initial condition $x(0) = \tilde{x}_0 \in \mathbb{R}^n$, where $\tilde{\phi}_i$ is a numerical solution in the i -th component to the system (3.5), and \tilde{x}_0 is located not far from x_0 , i.e., $\|x_0 - \tilde{x}_0\| \leq d$ for some small positive number $d \in \mathbb{R}$. We then define another mapping $\vec{\beta}$ as:

$$\vec{\beta}(\tilde{\phi}_i(kh)) := \begin{cases} \alpha(\phi_i(b)) & \text{if } \exists a, b \in \text{dom}(\phi_i; mq, h) \text{ s.t. } \phi_i(a) < \tilde{\phi}_i(kh) \leq \phi_i(b) \\ & \text{and } \nexists c \in \text{dom}(\phi_i; mq, h) \text{ s.t. } \phi_i(a) < \phi_i(c) \leq \phi_i(b), \\ \alpha(\min\{\phi_i(t)\}) & \text{if } \tilde{\phi}_i(kh) < \phi_i(t) \text{ for all } t \in \text{dom } \phi_i \\ \alpha(\max\{\phi_i(t)\}) & \text{if } \phi_i(t) < \tilde{\phi}_i(kh) \text{ for all } t \in \text{dom } \phi_i, \end{cases} \quad (3.7)$$

where $\text{dom}(\phi_i; mq, h) = \{t \in \text{dom } \phi_i : t \leq (mq-1)h\}$. Resulting in the sequence $\{\vec{\beta}(\tilde{\phi}_i(kh))\}_{k=0}^{mq-1}$, which represents a new variation of the original musical pitch. Additionally, we define the mapping $\overleftarrow{\beta}$ for an alternative way to create a new variation as:

$$\overleftarrow{\beta}(\tilde{\phi}_i(kh)) := \begin{cases} \alpha(\phi_i(a)) & \text{if } \exists a, b \in \text{dom}(\phi_i; mq, h) \text{ s.t. } \phi_i(a) \leq \tilde{\phi}_i(kh) < \phi_i(b) \\ & \text{and } \nexists c \in \text{dom}(\phi_i; mq, h) \text{ s.t. } \phi_i(a) \leq \phi_i(c) < \phi_i(b), \\ \alpha(\min\{\phi_i(t)\}) & \text{if } \tilde{\phi}_i(kh) < \phi_i(t) \text{ for all } t \in \text{dom}(\phi_i; mq, h), \\ \alpha(\max\{\phi_i(t)\}) & \text{if } \phi_i(t) < \tilde{\phi}_i(kh) \text{ for all } t \in \text{dom}(\phi_i; mq, h). \end{cases} \quad (3.8)$$

Resulting in the sequence $\{\overleftarrow{\beta}(\tilde{\phi}_i(kh))\}_{k=0}^{mq-1}$, which this mapping can use instead of $\vec{\beta}$.

In the final step, we construct a new variation that preserves the same duration of the musical pitch as in the original variation. Given the sequence

$$\{\vec{\beta}(\tilde{\phi}_i(kh))\}_{k=0}^{mq-1} = \{p_0^*, p_1^*, \dots, p_{mq-1}^*\},$$

we obtain a new sequence by selecting only the elements indexed by multiples of q , specifically $p_{k^*q}^*$, where $k^* \in \mathbb{N}_{m-1}$. This selection process results in the following sequence of musical pitch:

$$\{p_{k^*q}^*\}_{k^*=0}^{m-1} = \{p_0^*, p_q^*, p_{2q}^*, \dots, p_{(m-1)q}^*\}.$$

Similarly, we may use the sequence $\{\overleftarrow{\beta}(\tilde{\phi}_i(kh))\}_{k=0}^{mq-1}$ instead of $\{\vec{\beta}(\tilde{\phi}_i(kh))\}_{k=0}^{mq-1}$ to construct a new variation that likewise preserves the original musical pitch duration.

Example 3.3. Consider the first two bars of the sheet music for the music “Ah vous dirai-je Maman”, illustrated in Figure 3.8a. The sequence

$$\{p_k\}_{k=0}^6 = \{C4, C4, G4, G4, A4, A4, G4\}$$

represents the pitches in this excerpt, and

$$\{p'_k\}_{k=0}^{13} = \{C4, C4, C4, C4, G4, G4, G4, G4, A4, A4, A4, A4, G4, G4\}$$

represents the expanded rhythm pitches where $q = 2$, meaning each p_k is expanded rhythmically to 2 musical notes. Let

$$\dot{x}(t) = f(t, x) = \begin{bmatrix} f_1(t, x) \\ f_2(t, x) \\ f_3(t, x) \end{bmatrix} := \begin{bmatrix} 10(x_2 - x_1) \\ 28x_1 - x_2 - x_1x_3 \\ x_1x_2 - 2.6667x_3 \end{bmatrix} \quad (3.9)$$

where $x = (x_1, x_2, x_3) \in \mathbb{R}^3$. Firstly, we consider the numerical solution in the first component to the system (3.9) $\phi_1(t)$, derived using the fourth-order Runge-Kutta method with an initial

condition $x(0) = (1, 1, 1)$ and a step size $h = 0.01$. The first fourteen values of ϕ_1 are represented by the sequence

$$\begin{aligned}\{\phi_1(kh)\}_{k=0}^{13} &= \{\phi_1(0), \phi_1(0.01), \phi_1(0.02), \phi_1(0.03), \phi_1(0.04), \phi_1(0.05), \phi_1(0.06), \phi_1(0.07), \\ &\quad \phi_1(0.08), \phi_1(0.09), \phi_1(0.1), \phi_1(0.11), \phi_1(0.12), \phi_1(0.13)\} \\ &= \{1.00, 1.29, 2.13, 3.74, 6.54, 11.04, 16.69, 19.56, 15.37, 7.55, 1.20, \\ &\quad -2.67, -4.83, -6.12\}.\end{aligned}$$

We therefore define a mapping α as follows:

$$\alpha(\phi_1(kh)) := p_k.$$

For instance, the first element in $\{\phi_1(kh)\}_{k=0}^{13}$ is mapped to the first element in $\{p_k\}_{k=0}^{13}$, the second element in $\{\phi_1(kh)\}_{k=0}^{13}$ is mapped to the second element in $\{p_k\}_{k=0}^{13}$. Following this mapping, we obtain:

$$\begin{aligned}\alpha(\phi_1(0)) &= \alpha(1.00) = C4, \\ \alpha(\phi_1(0.01)) &= \alpha(1.29) = C4, \\ \alpha(\phi_1(0.02)) &= \alpha(2.13) = C4, \\ \alpha(\phi_1(0.03)) &= \alpha(3.74) = C4, \\ \alpha(\phi_1(0.04)) &= \alpha(6.54) = G4, \\ \alpha(\phi_1(0.05)) &= \alpha(11.04) = G4, \\ \alpha(\phi_1(0.06)) &= \alpha(16.69) = G4, \\ \alpha(\phi_1(0.07)) &= \alpha(19.56) = G4, \\ \alpha(\phi_1(0.08)) &= \alpha(15.37) = A4, \\ \alpha(\phi_1(0.09)) &= \alpha(7.55) = A4, \\ \alpha(\phi_1(0.10)) &= \alpha(1.20) = A4, \\ \alpha(\phi_1(0.11)) &= \alpha(-2.67) = A4, \\ \alpha(\phi_1(0.12)) &= \alpha(-4.83) = G4, \\ \alpha(\phi_1(0.13)) &= \alpha(-6.12) = G4,\end{aligned}$$

A summary of these mappings is provided in Table 3.2, while the overall mapping process of α is illustrated in Figure 3.6.

Next, we consider the numerical solution in the first component to the system (3.9) $\tilde{\phi}_1(t)$, derived again using the fourth-order Runge-Kutta method with the same step size h , but with a different initial condition $x(0) = (1.01, 1, 1)$. The first fourteen values of $\tilde{\phi}_1$ is represented as

$$\begin{aligned}\{\tilde{\phi}_1(kh)\}_{k=0}^{13} &= \{\tilde{\phi}_1(0), \tilde{\phi}_1(0.01), \tilde{\phi}_1(0.02), \tilde{\phi}_1(0.03), \tilde{\phi}_1(0.04), \tilde{\phi}_1(0.05), \tilde{\phi}_1(0.06), \tilde{\phi}_1(0.07), \\ &\quad \tilde{\phi}_1(0.08), \tilde{\phi}_1(0.09), \tilde{\phi}_1(0.1), \tilde{\phi}_1(0.11), \tilde{\phi}_1(0.12), \tilde{\phi}_1(0.13)\} \\ &= \{1.01, 1.30, 2.15, 3.76, 6.58, 11.10, 16.73, 19.55, 15.30, 7.48, 1.15 \\ &\quad -2.70, -4.85, -6.13\}.\end{aligned}$$

Let $\vec{\beta}$ be a mapping defined as (3.3), we note that

$$\text{dom}(\phi_1; 14, 0.01) = \{0, 0.01, 0.02, \dots, 0.13\}.$$

To illustrate this mapping, consider the following cases:

1. For $\tilde{\phi}_1(0) = 1.01$, we observe that there exist $0, 0.1 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0) < \tilde{\phi}_1(0) \leq \phi_1(0.1),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0) < \phi_1(c) \leq \phi_1(0.1).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0)) = \vec{\beta}(1.01) = \alpha(\phi_1(0.1)) = \alpha(1.20) = A4.$$

2. For $\tilde{\phi}_1(0.01) = 1.30$, we observe that there exist $0.01, 0.02 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.01) < \tilde{\phi}_1(0.01) \leq \phi_1(0.02),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.01) < \phi_1(c) \leq \phi_1(0.02).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.01)) = \vec{\beta}(1.30) = \alpha(\phi_1(0.02)) = \alpha(2.13) = C4.$$

3. For $\tilde{\phi}_1(0.02) = 2.15$, we observe that there exist $0.02, 0.03 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.02) < \tilde{\phi}_1(0.02) \leq \phi_1(0.03),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.02) < \phi_1(c) \leq \phi_1(0.03).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.02)) = \vec{\beta}(2.15) = \alpha(\phi_1(0.03)) = \alpha(3.74) = C4.$$

4. For $\tilde{\phi}_1(0.03) = 3.76$, we observe that there exist $0.03, 0.04 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.03) < \tilde{\phi}_1(0.03) \leq \phi_1(0.04),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.03) < \phi_1(c) \leq \phi_1(0.04).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.03)) = \vec{\beta}(3.76) = \alpha(\phi_1(0.04)) = \alpha(6.54) = G4.$$

5. For $\tilde{\phi}_1(0.04) = 6.58$, we observe that there exist $0.04, 0.09 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.04) < \tilde{\phi}_1(0.04) \leq \phi_1(0.09),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.04) < \phi_1(c) \leq \phi_1(0.09).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.04)) = \vec{\beta}(\tilde{\phi}_1(6.58)) = \alpha(\phi_1(0.09)) = \alpha(7.55) = A4.$$

6. For $\tilde{\phi}_1(0.05) = 11.10$, we observe that there exist $0.05, 0.08 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.05) < \tilde{\phi}_1(0.05) \leq \phi_1(0.08),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.05) < \phi_1(c) \leq \phi_1(0.08).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.05)) = \vec{\beta}(11.10) = \alpha(\phi_1(0.08)) = \alpha(15.37) = A4.$$

7. For $\tilde{\phi}_1(0.06) = 16.73$, we observe that there exist $0.05, 0.06 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.05) < \tilde{\phi}_1(0.06) \leq \phi_1(0.06),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.05) < \phi_1(c) \leq \phi_1(0.06).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.06)) = \vec{\beta}(16.73) = \alpha(\phi_1(0.06)) = \alpha(16.69) = G4.$$

8. For $\tilde{\phi}_1(0.07) = 19.56$, we observe that there exist $0.06, 0.07 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.06) < \tilde{\phi}_1(0.07) \leq \phi_1(0.07),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.06) < \phi_1(c) \leq \phi_1(0.07).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.07)) = \vec{\beta}(19.55) = \alpha(\phi_1(0.07)) = \alpha(19.56) = G4.$$

9. For $\tilde{\phi}_1(0.08) = 15.30$, we observe that there exist $0.05, 0.06 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.05) < \tilde{\phi}_1(0.08) \leq \phi_1(0.06),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.05) < \phi_1(c) \leq \phi_1(0.06).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.08)) = \vec{\beta}(15.30) = \alpha(\phi_1(0.06)) = \alpha(16.69) = G4.$$

10. For $\tilde{\phi}_1(0.09) = 7.48$, we observe that there exist $0.04, 0.09 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.04) < \tilde{\phi}_1(0.09) \leq \phi_1(0.09),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.04) < \phi_1(c) \leq \phi_1(0.09).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.09)) = \vec{\beta}(7.48) = \alpha(\phi_1(0.09)) = \alpha(7.55) = A4.$$

11. For $\tilde{\phi}_1(0.1) = 1.15$, we observe that there exist $0, 0.1 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0) < \tilde{\phi}_1(0.1) \leq \phi_1(0.1),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0) < \phi_1(c) \leq \phi_1(0.1).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.1)) = \vec{\beta}(1.15) = \alpha(\phi_1(0.1)) = \alpha(1.20) = A4.$$

12. For $\tilde{\phi}_1(0.11) = -2.70$, we observe that there exist $0.12, 0.11 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.12) < \tilde{\phi}_1(0.11) \leq \phi_1(0.11),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.12) < \phi_1(c) \leq \phi_1(0.11).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.11)) = \vec{\beta}(-2.70) = \alpha(\phi_1(0.11)) = \alpha(-2.70) = A4.$$

13. For $\tilde{\phi}_1(0.12) = -4.85$, we observe that there exist $0.13, 0.12 \in \text{dom}(\phi_1; 14, 0.01)$ such that:

$$\phi_1(0.13) < \tilde{\phi}_1(0.12) \leq \phi_1(0.12),$$

and since there is no $c \in \text{dom}(\phi_1; 14, 0.01)$ such that

$$\phi_1(0.13) < \phi_1(c) \leq \phi_1(0.12).$$

This yield:

$$\vec{\beta}(\tilde{\phi}_1(0.12)) = \vec{\beta}(-4.85) = \alpha(\phi_1(0.12)) = \alpha(-4.83) = G4.$$

14. For $\tilde{\phi}_1(0.13) = -6.13$, we observe that

$$\tilde{\phi}_1(0.13) < \phi_1(t),$$

for all $t \in \text{dom}(\phi_1; 14, 0.01)$. Since $\min \{\phi_1(kh)\}_{k=0}^{13} = -6.12$, this yield:

$$\vec{\beta}(\tilde{\phi}_1(0.13)) = \vec{\beta}(-6.13) = \alpha(\min\{\phi_1(t)\}) = \alpha(-6.13) = G4.$$

The complete mapping is summarized as follows:

$$\begin{aligned}
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0)) &= \beta(1.01) = A4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.01)) &= \beta(1.30) = C4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.02)) &= \beta(2.15) = C4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.03)) &= \beta(3.76) = G4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.04)) &= \beta(6.58) = A4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.05)) &= \beta(11.10) = A4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.06)) &= \beta(16.73) = G4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.07)) &= \beta(19.55) = G4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.08)) &= \beta(15.30) = A4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.09)) &= \beta(7.48) = A4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.10)) &= \beta(1.15) = A4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.11)) &= \beta(-2.70) = A4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.12)) &= \beta(-4.85) = G4, \\
\overset{\rightarrow}{\beta}(\tilde{\phi}_1(0.13)) &= \beta(-6.13) = G4.
\end{aligned}$$

Additionally, a summary of these mappings is presented in Table 3.2, and the overall mapping process of $\overset{\rightarrow}{\beta}$ is illustrated in Figure 3.7. Consequently, the mapping $\overset{\rightarrow}{\beta}$ produces the sequence:

$$\{\overset{\rightarrow}{\beta}(\tilde{\phi}_1(kh))\}_{k=0}^{13} = \{A4, C4, C4, G4, A4, A4, G4, G4, A4, A4, A4, A4, G4, G4\},$$

as illustrated in Figure 3.8b. To generate a new variation while preserving the original duration of the musical pitch, we utilize $q = 2$. Consequently, from $\{\overset{\rightarrow}{\beta}(\tilde{\phi}_1(kh))\}_{k=0}^{13}$, the sequence of new musical pitches is obtained as follows:

$$\{p_{k^*q}^*\}_{k^*=0}^6 = \{A4, C4, A4, G4, A4, A4, G4\},$$

as illustrated in Figure 3.8c. Furthermore, a visualization of this method is presented in Figure 3.9.

The resulting sequence of musical pitches exhibits significant divergence from the original sequence, with noticeable alterations in pitch that are perceivable by the audience. Such pronounced changes render this method conducive to generating more enchanting musical variations.

k	0	1	2	3	4	5	6	7	8	...	13
kh	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	...	0.13
$\phi_1(kh)$	1.00	1.29	2.13	3.74	6.54	11.04	16.69	19.56	15.37	...	-6.12
$\alpha(\phi_1(kh))$	C4	C4	C4	C4	G4	G4	G4	G4	A4	...	G4
$\tilde{\phi}_1(kh)$	1.01	1.30	2.15	3.76	6.58	11.10	16.73	19.55	15.30	...	-6.13
$\overset{\rightarrow}{\beta}(\tilde{\phi}_1(kh))$	A4	C4	C4	G4	A4	A4	G4	G4	A4	...	G4

Table 3.2: The result of a mapping g from $\{\phi_1(kh)\}_{k=0}^{13}$ to $\{p_k\}_{k=0}^{13}$ and the result of a mapping l from $\{\tilde{\phi}_1(kh)\}_{k=0}^{13}$ to new expanded musical pitches.

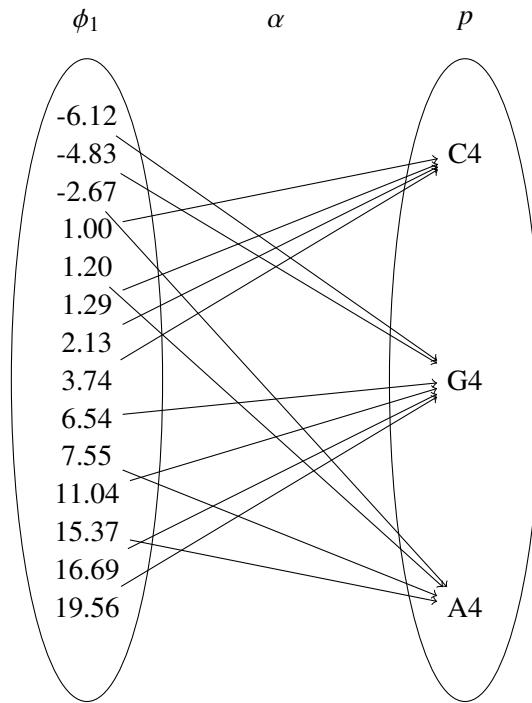


Figure 3.6: Mapping process of α

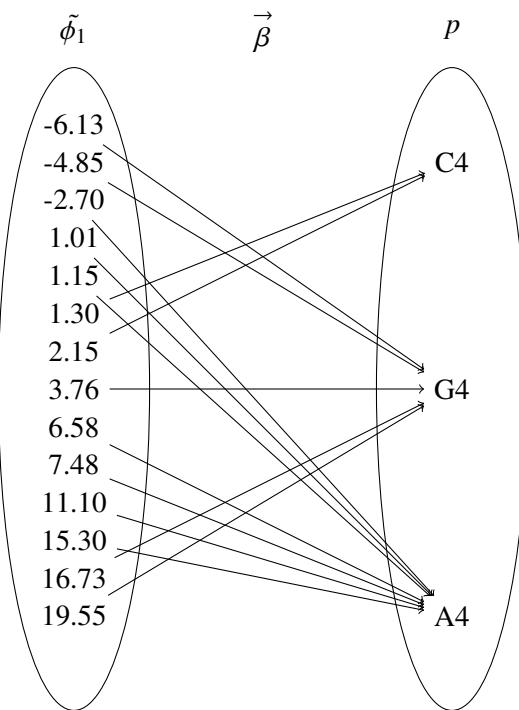


Figure 3.7: Mapping process of $\vec{\beta}$



(a) The original of Ah vous dirai-je Maman in the first 2 bars.

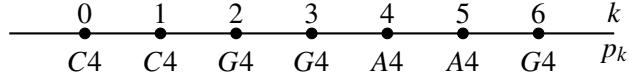


(b) The new expanded rhythm variation of Ah vous dirai-je, maman in the first 2 bar.

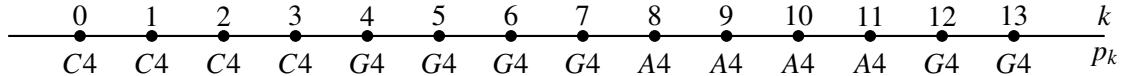


(c) The new variation of Ah vous dirai-je, maman in the first 2 bar.

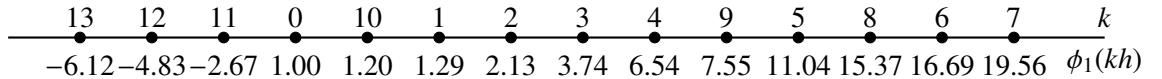
Figure 3.8: The original and musical variation from a expanded rhythm method of Ah vous dirai-je Maman.



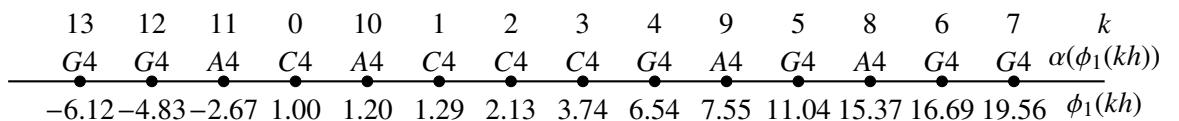
(a) The first 7 pitches of the Ah vous dirai-je Maman are marked below the 1D axis.



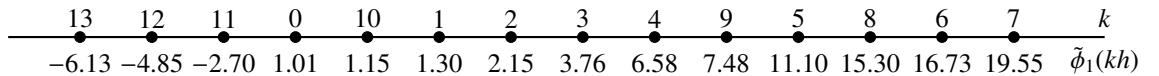
(b) The first 14 expanded rhythm pitches of the Ah vous dirai-je Maman are marked below the 1D axis.



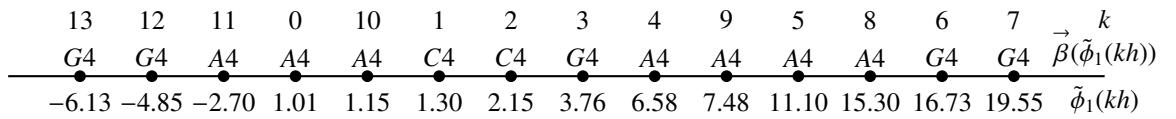
(c) The first 14 numerical solution in first component to the system (3.9) with initial condition of (1, 1, 1) are marked below the 1D axis.



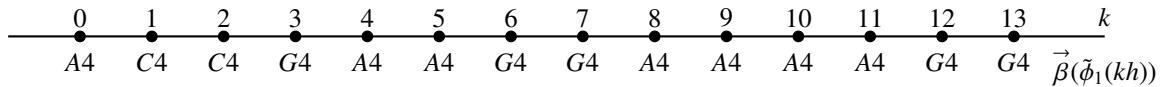
(d) For each component in $\{\phi_1(kh)\}_{k=0}^6$, apply the $\alpha(\phi_1(kh))$ mapping.



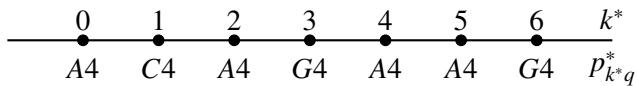
(e) The first 14 numerical solution in first component to the system (3.9) with new initial condition of (1.01, 1, 1) are marked below the 1D axis.



(f) For each component in $\{\tilde{\phi}_1(kh)\}_{k=0}^{13}$, apply the $\beta_tilde(\tilde{\phi}_1(kh))$ mapping.



(g) The new expanded rhythm variation of the first 14 pitches is marked below the 1D axis.



(h) The new variation of the first 7 pitches with the same duration as the original musical pitches.

Figure 3.9: The figure for visualizes how a chaotic mapping with musical variation method can be used to generate musical variations

Example 3.4. From Example 3.3, if we use the mapping $\overset{\leftarrow}{\beta}$ from $\{\tilde{\phi}_1(kh)\}_{k=0}^{13}$ to new musical pitches according to the mapping (3.8), we obtain the sequence:

$$\{\overset{\leftarrow}{\beta}(\tilde{\phi}_1(kh))\}_{k=0}^{13} = \{C4, C4, C4, C4, G4, G4, A4, G4, G4, G4, C4, G4, G4, G4\},$$

as illustrated in Figure 3.8b. To generate a new variation while preserving the original duration of the musical pitch, we utilize $q = 2$. Consequently, from $\{\overset{\rightarrow}{\beta}(\tilde{\phi}_1(kh))\}_{k=0}^{13}$, the sequence of new musical pitches is obtained as follows:

$$\{p_{k^*q}^*\}_{k^*=0}^6 = \{A4, C4, A4, G4, A4, A4, G4\},$$

as illustrated in Figure 3.10c. Furthermore, a visualization of this method is presented in Figure 3.11.

This example shows that changing the mapping from $\overset{\rightarrow}{\beta}$ to $\overset{\leftarrow}{\beta}$ can create different pitches for new variations. However, determining the best mapping for generating new variations depends on the listener's preference. We provide $\overset{\leftarrow}{\beta}$ as an alternative choice for creating new variations.



(a) The original of Ah vous dirai-je Maman in the first 2 bars.

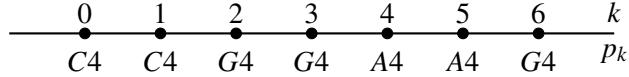


(b) The new expanded rhythm variation of Ah vous dirai-je, maman in the first 2 bar.

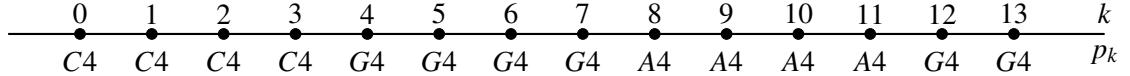


(c) The new variation of Ah vous dirai-je, maman in the first 2 bar.

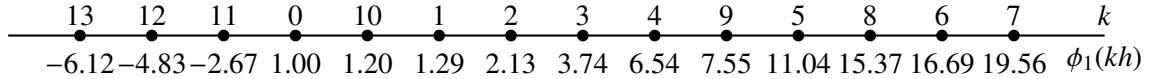
Figure 3.10: The original and musical variation from a expanded rhythm method of Ah vous dirai-je Maman with mapping $\overset{\leftarrow}{\beta}$.



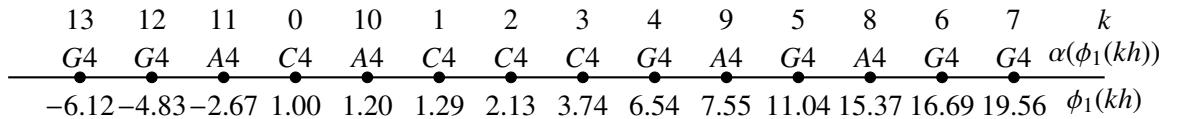
(a) The first 7 pitches of the Ah vous dirai-je Maman are marked below the 1D axis.



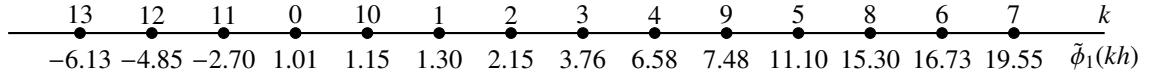
(b) The first 14 expanded rhythm pitches of the Ah vous dirai-je Maman are marked below the 1D axis.



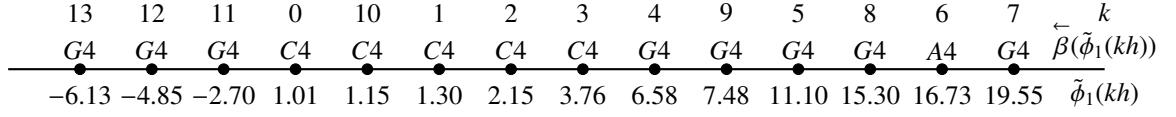
(c) The first 14 numerical solution in first component to the system (3.9) with initial condition of (1, 1, 1) are marked below the 1D axis.



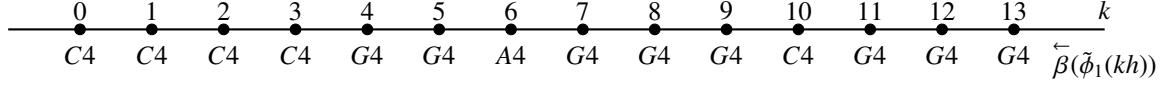
(d) For each component in $\{\phi_1(kh)\}_{k=0}^6$, apply the $\alpha(\phi_1(kh))$ mapping.



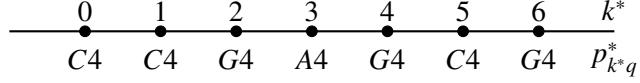
(e) The first 14 numerical solution in first component to the system (3.9) with new initial condition of (1.01, 1, 1) are marked below the 1D axis.



(f) For each component in $\{\tilde{\phi}_1(kh)\}_{k=0}^{13}$, apply the $\beta(\tilde{\phi}_1(kh))$ mapping.



(g) The new expanded rhythm variation of the first 14 pitches is marked below the 1D axis.



(h) The new variation of the first 7 pitches with the same duration as the original musical pitches.

Figure 3.11: Visualizes the result of using the mapping β .

3.4 Showcase

In this section, we present the results of applying the expanded rhythm method to existing sheet music from historical compositions. This serves to demonstrate the applicability of our method to pre-existing works and to illustrate the musical variations produced by our approach.

Example 3.5. Consider the sheet music of Pachelbel's Canon in D, as illustrated in Figure 3.12. Let the sequence $\{p_k\}$ denote the pitches in the Treble Clef (\textcircled{G}). Furthermore, let the sequence $\{p'_k\}$ represent the rhythmically expanded pitches, where each p_k is subdivided into four musical notes. Let

$$\dot{x}(t) = f(t, x) = \begin{bmatrix} f_1(t, x) \\ f_2(t, x) \\ f_3(t, x) \end{bmatrix} := \begin{bmatrix} 10(x_2 - x_1) \\ 28x_1 - x_2 - x_1x_3 \\ x_1x_2 - 2.6667x_3 \end{bmatrix} \quad (3.10)$$

where $x = (x_1, x_2, x_3) \in \mathbb{R}^3$. First, we compute the numerical solution for the first component of system (3.13), denoted as $\phi_1(t)$, using the fourth-order Runge-Kutta method with an initial condition $x(0) = (1.5, 1.5, 1.5)$ and a step size $h = 0.01$. The values of ϕ_1 form the sequence $\{\phi_1\}$. We define a mapping α as follows:

$$\alpha(\phi_1(kh)) := p_k.$$

Next, we compute the numerical solution for the first component of system (3.13), denoted as $\tilde{\phi}_1(t)$, using the same method and step size but with a different initial condition $x(0) = (1.51, 1.51, 1.51)$. The values of $\tilde{\phi}_1$ form the sequence $\{\tilde{\phi}_1\}$.

By applying musical variations through the expanded rhythm method utilizing the mapping $\overset{\rightarrow}{\beta}$, as defined in (3.7), the resulting variation is depicted in Figure 3.13 within the Treble Clef (\textcircled{G}). Conversely, when employing the expanded rhythm method with the mapping $\overset{\leftarrow}{\beta}$, as defined in (3.8), the corresponding variation is illustrated in Figure 3.14, also within the Treble Clef (\textcircled{G}).

Similarly, consider the sheet music of Pachelbel's Canon presented in Figure 3.12. The sequence $\{p_k\}$ represents the pitches in the Bass Clef (\textcircled{F}). Furthermore, let the sequence $\{p'_k\}$ represent the rhythmically expanded pitches, where each p_k is subdivided into two musical notes. Let

$$\dot{x}(t) = f(t, x) = \begin{bmatrix} f_1(t, x) \\ f_2(t, x) \\ f_3(t, x) \end{bmatrix} := \begin{bmatrix} 10(x_2 - x_1) \\ 28x_1 - x_2 - x_1x_3 \\ x_1x_2 - 2.6667x_3 \end{bmatrix} \quad (3.11)$$

where $x = (x_1, x_2, x_3) \in \mathbb{R}^3$. Following the same numerical approach, we determine $\phi_1(t)$ and $\tilde{\phi}_1(t)$ for the Bass Clef sequence. If we apply the expanded rhythm method with the mapping $\overset{\rightarrow}{\beta}$, as defined in (3.7), the resulting variation appears in Figure 3.13 in the Bass Clef (\textcircled{F}). Conversely, applying the mapping $\overset{\leftarrow}{\beta}$, as defined in (3.8), produces the variation depicted in Figure 3.14 in the Bass Clef (\textcircled{F}).

The complete musical score generated through this method is provided in Appendix Music Sheets.



Figure 3.12: First two bars of Pachelbel’s Canon.



Figure 3.13: New variation of Pachelbel’s Canon using the mapping β^+ in the first two bars.

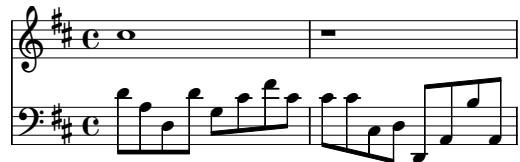


Figure 3.14: New variation of Pachelbel’s Canon using the mapping β^- in the first two bars.

Example 3.6. Consider the sheet music of River Flows In You, as illustrated in Figure 3.12. Let the sequence $\{p_k\}$ denote the pitches in the Treble Clef ($\textcircled{1}$). Furthermore, let the sequence $\{p'_k\}$ represent the rhythmically expanded pitches, where each p_k is subdivided into four musical notes. Let

$$\dot{x}(t) = f(t, x) = \begin{bmatrix} f_1(t, x) \\ f_2(t, x) \\ f_3(t, x) \end{bmatrix} := \begin{bmatrix} 10(x_2 - x_1) \\ 28x_1 - x_2 - x_1x_3 \\ x_1x_2 - 2.6667x_3 \end{bmatrix} \quad (3.12)$$

where $x = (x_1, x_2, x_3) \in \mathbb{R}^3$. First, we compute the numerical solution for the first component of system (3.13), denoted as $\phi_1(t)$, using the fourth-order Runge-Kutta method with an initial condition $x(0) = (1.5, 1.5, 1.5)$ and a step size $h = 0.01$. The values of ϕ_1 form the sequence $\{\phi_1\}$. We define a mapping α as follows:

$$\alpha(\phi_1(kh)) := p_k.$$

Next, we compute the numerical solution for the first component of system (3.13), denoted as $\tilde{\phi}_1(t)$, using the same method and step size but with a different initial condition $x(0) = (1.51, 1.51, 1.51)$. The values of $\tilde{\phi}_1$ form the sequence $\{\tilde{\phi}_1\}$.

By applying musical variations through the expanded rhythm method utilizing the mapping β^+ , as defined in (3.7), the resulting variation is depicted in Figure 3.16 within the Treble Clef ($\textcircled{1}$). Conversely, when employing the expanded rhythm method with the mapping β^- , as defined in (3.8), the corresponding variation is illustrated in Figure 3.17, also within the Treble Clef ($\textcircled{1}$).

Similarly, consider the sheet music of River Flows In You presented in Figure 3.15. The sequence $\{p_k\}$ represents the pitches in the Bass Clef ($\textcircled{2}$). Furthermore, let the sequence

$\{p'_k\}$ represent the rhythmically expanded pitches, where each p_k is subdivided into two musical notes. Let

$$\dot{x}(t) = f(t, x) = \begin{bmatrix} f_1(t, x) \\ f_2(t, x) \\ f_3(t, x) \end{bmatrix} := \begin{bmatrix} 10(x_2 - x_1) \\ 28x_1 - x_2 - x_1 x_3 \\ x_1 x_2 - 2.6667 x_3 \end{bmatrix} \quad (3.13)$$

where $x = (x_1, x_2, x_3) \in \mathbb{R}^3$. Following the same numerical approach, we determine $\phi_1(t)$ and $\tilde{\phi}_1(t)$ for the Bass Clef sequence. If we apply the expanded rhythm method with the mapping $\vec{\beta}$, as defined in (3.7), the resulting variation appears in Figure 3.16 in the Bass Clef (\mathfrak{B}). Conversely, applying the mapping $\overset{\leftarrow}{\beta}$, as defined in (3.8), produces the variation depicted in Figure 3.17 in the Bass Clef (\mathfrak{B}).

The complete musical score generated through this method is provided in Appendix Showcase Full Music Sheet.



Figure 3.15: First two bars of River Flows In You.



Figure 3.16: New variation of River Flows In You using the mapping $\vec{\beta}$ in the first two bars.



Figure 3.17: New variation of River Flows In You using the mapping $\overset{\leftarrow}{\beta}$ in the first two bars.

The results indicate that the newly generated variations contain a significantly greater number of musical notes compared to the original composition, leading to an increased tempo. Furthermore, the two variations exhibit distinct musical characteristics. The variation produced using $\overset{\leftarrow}{\beta}$ retains a strong resemblance to the original melody while accelerating the tempo and incorporating slight modifications in note placement. In contrast, the variation derived from $\vec{\beta}$ introduces a novel pattern of musical notes, resulting in a melody that diverges more noticeably from the original composition.

These findings suggest that this method can serve as a valuable tool for musical composition, providing new possibilities for melodic transformation and creative expression.

Chapter 4

Discussion and Conclusion

This chapter provides a comprehensive discussion of both the Python library and the web application, outlining their key benefits and usage. Furthermore, a summary of the project is presented to encapsulate its contributions and potential applications.

4.1 Discussion

This section elaborates on the advantages, installation process, and usage of both the Python library and the web application. This information is crucial for researchers, students, and developers who are interested in leveraging our algorithm for music generation or extending its capabilities for generating musical variations.

4.1.1 Python Library

The Python library is designed as a foundational tool for students and researchers engaged in this project. It serves as a valuable resource for studying the process of generating musical variations and facilitating further development. The library is structured with simplicity in mind, ensuring accessibility and ease of understanding.

One of the primary advantages of this Python library is its flexibility in allowing users to customize function settings according to their specific requirements. Users can modify parameters, adjust function configurations, and extend existing functions to enhance the library's functionality. Moreover, new functions can be developed to improve the efficiency and scope of musical variation generation.

However, a notable limitation of this Python library is its prerequisite programming knowledge. Users are required to have proficiency in Python and an understanding of the library's function usage. Given that all interactions are conducted through code, individuals without prior programming experience may find the library challenging to use.

1) Installation

To install the Python library, it is recommended to use Git, which can be downloaded from <https://git-scm.com/downloads>. The installation process involves the following steps:

1. Create a new directory for the project.
2. Open a command prompt within the newly created directory.
3. Execute the following command to clone the repository:

```
C:\Users\YourFolderLocation> git clone  
→ https://github.com/Ataetano/project_application.git
```

Upon successful execution, the command prompt will display an output similar to the following:

```

Cloning into 'project_application'...
remote: Enumerating objects: 1289, done.
remote: Counting objects: 100% (1289/1289), done.
remote: Compressing objects: 100% (817/817), done.
remote: Total 1289 (delta 403), reused 1289 (delta 403), pack-reused 0 (from
→ 0)
Receiving objects: 100% (1289/1289), 36.68 MiB | 572.00 KiB/s, done.
Resolving deltas: 100% (403/403), done.
Updating files: 100% (1145/1145), done.

```

Next, Python must be installed, which can be downloaded from <https://www.python.org/downloads/>. After installing Python, the required dependencies should be installed by executing the following commands in the command prompt:

```
pip install -r requirements.txt
```

Upon successful installation of all required packages, the Python library will be ready for use. The library is located within the `project_application` directory, specifically in the `python_library` subdirectory created during the cloning process. The essential files within the `python_library` folder include `melody.py` and `lilypond-x.xx.x`.

To facilitate efficient use of the library, it is recommended to utilize Jupyter Notebook or any other suitable Python development environment. The library can be initialized using the following code within the `python_library` directory:

```

1 from melody import Melody
2 atb = Melody()

```

This completes the setup process. It is important to ensure that all required files are present within the `python_library` folder. The script utilizing this library should be located in the same directory as `melody.py` and `lilypond-x.xx.x`.

Tutorial for Generating Musical Variations

This section outlines the standard methodology for generating musical variations using our Python library. The provided examples can be seamlessly integrated into research projects or practical applications. Throughout this tutorial, we assume the existence of a MIDI file named `Canon_in_D.mid`, located in the main directory. Additionally, our Python library includes an implementation of the Lorenz system, modeled as a dynamical system. The corresponding Python code is provided as follows:

```

1 def lorenz_system(t, x, sigma=10, rho=28, beta=8/3):
2     """
3         Computes the derivatives of the Lorenz system at a given time t.
4         Parameters:
5             t (int or float): Time variable (not explicitly used in computation).
6             x (array_like): State vector at time t, given as a list or NumPy array
7                 → [x_1, x_2, x_3].
8             sigma (float): Lorenz system parameter (must be greater than 0).
9             rho (float): Lorenz system parameter (must be greater than 0).
10            beta (float): Lorenz system parameter (must be greater than 0).
11
12        Returns:
13            numpy.ndarray: The new state vector [x_1dot, x_2dot, x_3dot] as a NumPy
14                → array.
15        """

```

```

14     x_1, x_2, x_3 = x
15     x_1dot = sigma * (x_2 - x_1)
16     x_2dot = x_1 * (rho - x_3) - x_2
17     x_3dot = x_1 * x_2 - beta * x_3
18     return np.array([x_1dot, x_2dot, x_3dot])

```

Example 4.1. In this example, we illustrate the standard usage of our Python package by following these steps. First, we import the necessary module and instantiate a `Melody` object:

```

import numpy as np
from melody import Melody
atb = Melody()

```

Next, we load the MIDI file `Canon_in_D.mid` into the Python script:

```
atb.load("Canon_in_D.mid")
```

To generate a numerical solution and map each element of the solution to a musical pitch within the MIDI file using the mapping α , execute the following command:

```
atb.fit(np.array([1.5, 1.5, 1.5]))
```

Here, the NumPy array `np.array([1.5, 1.5, 1.5])` defines the initial condition $x(0) = x_0$ for the numerical solution generation. Subsequently, a new numerical solution can be derived and mapped to a new musical pitch sequence using the mapping β as follows:

```
atb.variate(np.array([1.51, 1.51, 1.51]))
```

Here, the NumPy array `np.array([1.51, 1.51, 1.51])` represents the new initial condition $x(0) = \tilde{x}_0$ for generating the new numerical solution. Finally, to export the newly generated musical variation in both MIDI and PDF formats, execute:

```

atb.export("midi")
atb.export("pdf")

```

The complete workflow is summarized in the following script:

```

1 import numpy as np
2 from melody import Melody
3
4 atb = Melody()
5 atb.load("Canon_in_D.mid")
6 atb.fit(np.array([1.5, 1.5, 1.5]))
7 atb.variate(np.array([1.51, 1.51, 1.51]))
8 atb.export("midi")
9 atb.export("pdf")

```

Example 4.2. In this example, we demonstrate the advanced usage of our Python package by following a structured approach. First, we import the necessary module and instantiate a `Melody` object:

```

import numpy as np
from melody import Melody
atb = Melody()

```

Next, we illustrate how to define a custom dynamical system, should the default Lorenz system not be preferred. This requires implementing the dynamical system in a Python script as follows:

```

1 def rossler_system(t, x, a, b, c):
2     """
3         Computes the derivatives of the Rössler system at a given time t.
4
5     Parameters:
6         t (int or float): Time variable (not explicitly used in computation).
7         x (array_like): State vector at time t, given as a list or NumPy array
8             ↳ [x_1, x_2, x_3].
9         a (float): Rössler system parameter (must be greater than 0).
10        b (float): Rössler system parameter (must be greater than 0).
11        c (float): Rössler system parameter (must be greater than 0).
12
13    Returns:
14        numpy.ndarray: The new state vector [x_1dot, x_2dot, x_3dot] as a NumPy
15            ↳ array.
16        """
17        x_1, x_2, x_3 = x
18        x_1dot = -x_2 - x_3
19        x_2dot = x_1 + a * x_2
20        x_3dot = b + x_3 * (x_1 - c)
21        return np.array([x_1dot, x_2dot, x_3dot])

```

Once the custom dynamical system has been defined, it can be imported and its parameters set using the following code:

```
atb.set_dynamic(rossler_system, a=0.2, b=0.2, c=5.7)
```

To select the numerical solution for a specific component x_1 of the dynamical system, the following command can be used:

```
atb.set_dynamic_sequence(1)
```

Here, the argument 1 corresponds to the component x_1 . Similarly, setting this argument to 2 or 3 selects the components x_2 or x_3 , respectively. To generate a numerical solution, map each element of the solution to a musical pitch within the MIDI file using the mapping α . The following command executes this step:

```
atb.fit(np.array([1.5, 1.5, 1.5]), method="expand", divisions=4)
```

In this method, `np.array([1.5, 1.5, 1.5])` defines the initial condition $x(0) = x_0$ for numerical solution generation. The argument `method="expand"` specifies that the musical variations will be generated using an expanded rhythm approach, while `divisions=4` expands each musical pitch in the MIDI file into four musical notes. If the chaotic mapping method is preferred, the arguments `method="expand"` and `divisions=4` should be omitted, as the system defaults to this approach. After executing `atb.fit`, the available tracks within the MIDI file can be verified using:

```
atb.show_all_track()
```

This produces the output:

```
dict_keys(['Piano_0', 'track_1'])
```

Subsequently, a new numerical solution can be generated, mapped to a new musical pitch sequence using the mapping β , and applied to a specific track. This is achieved as follows:

```
atb.variate(np.array([1.51, 1.51, 1.51]), track=['Piano_0'], add_note=50)
```

Here, `np.array([1.51, 1.51, 1.51])` represents the new initial condition $x(0) = \tilde{x}_0$. The argument `track=['Piano_0']` specifies that the variation is applied solely to the `Piano_0` track, while `add_note=50` denotes the addition of 50 musical notes to the new variation. Finally, the generated musical variation can be exported in both MIDI and PDF formats using:

```
atb.export("midi")
atb.export("pdf")
```

The complete workflow is summarized in the following script:

```

1 import numpy as np
2 from melody import Melody
3
4 def rossler_system(t, x, a, b, c):
5     """
6         Computes the derivatives of the Rössler system at a given time t.
7
8     Parameters:
9         t (int or float): Time variable (not explicitly used in computation).
10        x (array_like): State vector at time t, given as a list or NumPy array
11            ↳ [x_1, x_2, x_3].
12        a (float): Rössler system parameter (must be greater than 0).
13        b (float): Rössler system parameter (must be greater than 0).
14        c (float): Rössler system parameter (must be greater than 0).
15
16    Returns:
17        numpy.ndarray: The new state vector [x_1dot, x_2dot, x_3dot] as a NumPy
18            ↳ array.
19        """
20        x_1, x_2, x_3 = x
21        x_1dot = -x_2 - x_3
22        x_2dot = x_1 + a * x_2
23        x_3dot = b + x_3 * (x_1 - c)
24        return np.array([x_1dot, x_2dot, x_3dot])
25
26 atb = Melody()
27 atb.load("Canon_in_D.mid")
28 atb.set_dynamic(rossler_system, a=0.2, b=0.2, c=5.7)
29 atb.set_dynamic_sequence(1)
30 atb.fit(np.array([1.5, 1.5, 1.5]), method="expand", divisions=4)
31 atb.variate(np.array([1.51, 1.51, 1.51]), track=['Piano_0'], add_note=50)
32 atb.export("midi")
33 atb.export("pdf")
```

3) Library Class and Methods

The Python library includes a primary class, `Melody`, which is responsible for processing the algorithm. This class contains all the necessary code for generating musical variations. To effectively utilize the library, it is essential to understand the instance attributes and available methods provided within this class.

3.1) Instance Attributes

This topic outlines the instance attributes of the `Melody` class, which are fundamental for the process of generating musical variations. These attributes are defined in the library as follows:

```
1 def __init__(self):
2     self.tracks = {}
3     self.maps = {}
4     self.dynamic = self.lorenz_system
5
6     # Main Instance Attributes
7     self.original_midi_data = None
8     self.new_midi_data = None
9
10    # Additional Instance Attributes
11    self.newtracks = {}
12    self.main_stream = None
13    self.new_stream = None
14    self.duration = {}
15    self.new_duration = {}
16    self.original_initial_condition = None
17    self.new_initial_condition = None
18    self.dynamic_sequence = []
19    self.notetrack = {}
20    self.tracks_index = {}
21    self.dynamic_parameter = {}
22
23    # Support Instance Attributes
24    self.method = None
25    self.divisions = None
26    self.backtrack_expand_index = []
27
28    # Path
29    self.lilypond_path = str(pathlib.Path("__file__").parent.resolve()) +
30    ↵ "\\\lilypond-2.24.4\\bin\\lilypond.exe"
31
32    # etc.
33    self.filename = None
```

To clarify the purpose of each instance attribute, the following list provides detailed explanations:

Melody.tracks (dict): Stores track names as keys and the corresponding trajectory sequences as values.

Melody.maps (dict): Stores track names as keys and a list containing trajectory sequences and musical note sequences of the original variation as values.

Melody.dynamic (function): Represents the dynamical system as a Python function. By default, the Lorenz system is predefined in the library.

Melody.original_midi_data (music21_object): Stores MIDI file data, serving as the original variation of the music.

Melody.new_midi_data (music21_object): Stores MIDI file data used for generating new musical variations while maintaining the same note durations as the original variation.

- Melody.newtracks (dict)**: Stores track names as keys and the corresponding musical sequences of the new variation as values.
- Melody.main_stream (music21_object)**: Stores MIDI file data, serving as the original version of the music for use in computational processes.
- Melody.new_stream (music21_object)**: Stores MIDI file data used for generating new expanded rhythm musical variations.
- Melody.duration (dict)**: Stores track names as keys and the musical note duration sequences of the original variation as values.
- Melody.new_duration (dict)**: Stores track names as keys and the musical note duration sequences of the new variation as values.
- Melody.original_initial_condition (array_like)**: Stores a NumPy array representing the initial conditions of the dynamical system used for generating trajectories of the original variation.
- Melody.new_initial_condition (array_like)**: Stores a NumPy array representing the initial conditions of the dynamical system used for generating trajectories of the new variation.
- Melody.dynamic_sequence (int)**: Stores an integer representing the selected trajectory sequence variable.
- Melody.notetrack (dict)**: Stores track names as keys and the corresponding musical note sequences of the original variation as values.
- Melody.tracks_index (dict)**: Stores track names as keys and their corresponding index values, which are used to select specific tracks for generating new variations.
- Melody.dynamic_parameter (dict)**: Stores parameter names as keys and their corresponding values.
- Melody.method (str)**: Stores the method name used for generating the new variation.
- Melody.divisions (int)**: Represents the number of subdivisions for the expanded rhythm of each musical note in the MIDI file.
- Melody.backtrack_expand_index (list)**: Stores a list that maps the musical pitches from the expanded rhythmic sequence to a new variation while preserving the original note durations.
- Melody.lilypond_path (str)**: Stores the file path to the LilyPond program, which is used for generating PDF music scores.
- Melody.filename (str)**: Stores the name of the MIDI file.

3.2) Available Methods

This topic focuses on the primary method of the `Melody` class. For additional methods not covered here, refer to Appendix **Source Code**. Before utilizing any method, it is essential to ensure that the following initialization code is included in your script:

```

1 import numpy as np
2 from melody import Melody
3 atb = Melody()

```

3.2.1) Load Method

```
Melody.load(path)
```

Load a MIDI file.

Parameter:

path : str

The path to the MIDI file on your computer. The file extension must be of type `.mid`.

Return:

None

Example:

In this example, we assume that you have a MIDI file named `Canon_in_D.mid` stored on your computer. You can load this MIDI file using the full path as shown below:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load(r"C:\Users\YourFolderLocation\Canon_in_D.mid")
```

Alternatively, if the MIDI file is in the same location as `melody.py` and `lilypond-x.xx.x`, you can use:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load("Canon_in_D.mid")
```

to load the MIDI file without specifying the full path, we assume that all examples will use a MIDI file named `Canon_in_D.mid`, stored in the main folder.

3.2.2) Choosing Dynamical System Method (Optional)

```
Melody.set_dynamic(dynamic, *)
```

Import a dynamical system and parameter values for generating the trajectory. This function is optional if you want to use the Lorenz system as the dynamical system, which is already available in this library.

Parameter:

dynamic : function

A user-defined function where:

- The input time (`t`) is of type `int` or `float`.
- The initial condition is a `numpy` array or array like.
- Parameters are of type `int` or `float`.
- The output is a `numpy` array by default.

**kwargs

Parameter values for the dynamical system, which depend on the defined dynamical system function. By default, this function can accept `sigma`, `rho`, and `beta` for setting Lorenz system parameters.

Return:

`None`

Example:

In this example, we assume that you define a function named `rossler_system` as follows:

```
1 def rossler_system(t, x, a, b, c):
2     x_1, x_2, x_3 = x
3     x_1dot = -x_2 - x_3
4     x_2dot = x_1 + a * x_2
5     x_3dot = b + x_3 * (x_1 - c)
6     return np.array([x_1dot, x_2dot, x_3dot])
```

You can then set this function as the dynamical system and parameter values using the following code:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.set_dynamic(rossler_system, a=0.2, b=0.2, c=5.7)
```

3.2.3) Dynamical System Sequence Method (Optional)

`Melody.set_dynamic_sequence(sequence)`

Sets the sequence of the numerical solution from the dynamical system, which corresponds to $\tilde{\phi}_i$ from Section 3.1. This function is optional if you wish to use the sequence x_1 from the Lorenz system as the numerical solution, which is already available in this library.

Parameters:

`sequence : int`

An integer representing the selected sequence of the numerical solution from the dynamical system. By default, 1 refers to sequence x_1 , 2 refers to sequence x_2 , and 3 refers to sequence x_3 of the Lorenz system.

Return:

`None`

Example:

In this example, we assume that you define a function named `rossler_system` as follows:

```
1 def rossler_system(t, x, a, b, c):
2     x_1, x_2, x_3 = x
3     x_1dot = -x_2 - x_3
4     x_2dot = x_1 + a * x_2
5     x_3dot = b + x_3 * (x_1 - c)
6     return np.array([x_1dot, x_2dot, x_3dot])
```

You can then set the sequence you want to be the main numerical solution using the following code:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.set_dynamic(lorenz_system)
5 >>> atb.set_dynamic_sequence(1)
```

Alternatively, if you do not define a function, you can use the following code to set the sequence of the available Lorenz system in this library:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.set_dynamic_sequence(1)
```

3.2.4) Trajectory Method

```
Melody.fit(original_initial_condition, method="classic",
           divisions=None):
```

Constructs the trajectory for the original variation from a MIDI file, sets the method for generating a new variation, and maps the trajectory to musical pitches using α .

Parameters:

original_initial_condition : array_like

Initial condition of the original variation trajectory as a NumPy array.

method : str

Method for generating a new variation. Use "classic" to create a variation using musical variations from a chaotic mapping method and "expand" for using musical variation from an expanded rhythm method.

divisions : int

The number of subdivisions for expanding each musical note from the original. Available choices: 2, 4, 8, 16, 32.

Return:

None

Example:

In this example, we consider the case where you did not define a function for the dynamical system. You can create the trajectory of this dynamical system with the following code:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load("Canon_in_D.mid")
5 >>> atb.fit(np.array([1.5, 1.5, 1.5]))
```

Alternatively, if you want to use a different method, you can use the following code:

```

1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load("Canon_in_D.mid")
5 >>> atb.fit(np.array([1.5, 1.5, 1.5]), method="expand", divisions=2)

```

3.2.5) Track Method

`Melody.show_all_track()`

Print all of the tracks in the MIDI file, which will be available after using the function `atb.fit`.

Example:

In this example, we consider the case where you did not define a function for the dynamical system. You can check the track name by using the following code:

```

1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load("Canon_in_D.mid")
5 >>> atb.fit(np.array([1.5, 1.5, 1.5]))
6 >>> atb.show_all_track()
7 dict_keys(['Piano_0', 'track_1'])

```

3.2.6) Variation Method

`Melody.variate(new_initial_condition, track=None,
 ↵ criteria="right", add_note=0)`

Creating the trajectory for the new expanded rhythm variation from a MIDI file and denoting the new musical note using the mapping $\overset{\rightarrow}{\beta}$ or $\overset{\leftarrow}{\beta}$ from Subsection 3.3.

Parameters:

`new_initial_condition : array_like`

Initial condition of the new variation trajectory as a NumPy array.

`track : list`

List of track names for generating a new variation. If `track` is `None`, this function will generate a new variation for every track in the MIDI file.

`criteria : str`

Select the mapping for creating a new variation. Use "right" for mapping $\overset{\rightarrow}{\beta}$ and "left" for mapping $\overset{\leftarrow}{\beta}$.

`add_note : int`

The number of musical notes to add after the end of the MIDI file.

Return:

`None`

Example:

In this example, we consider the case where you did not define a function for the dynamical system. You can generate the trajectory for the new variation using the following code:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load("Canon_in_D.mid")
5 >>> atb.fit(np.array([1.5, 1.5, 1.5]), method="expand", divisions=2)
6 >>> atb.show_all_track()
7 dict_keys(['Piano_0', 'track_1'])
8 >>> atb.variate(np.array([1.51, 1.51, 1.51]), track=['Piano_0'],
   ↵ criteria="right")
```

Or, if you want to generate the new variation for all tracks, use this code instead:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load("Canon_in_D.mid")
5 >>> atb.fit(np.array([1.5, 1.5, 1.5]), method="expand", divisions=2)
6 >>> atb.variate(np.array([1.51, 1.51, 1.51]), criteria="right")
```

3.2.7) Export Method

`Melody.export(format_type)`

Creates MIDI files of the original and new variations or generates music sheets of the original and new variations in your folder.

Parameters:

`format_type : str`

Use "midi" to generate MIDI files and "pdf" to generate music sheet files in PDF format.

Return:

`None`

Example:

In this example, we consider the case where you did not define a function for the dynamical system. To generate the MIDI file of the original and new variation, use the following code:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load("Canon_in_D.mid")
5 >>> atb.fit(np.array([1.5, 1.5, 1.5]), method="expand", divisions=2)
6 >>> atb.variate(np.array([1.51, 1.51, 1.51]), criteria="right")
7 >>> atb.export("midi")
```

Or, if you want to generate the music sheets of the original and new variation, use this code instead:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load("Canon_in_D.mid")
5 >>> atb.fit(np.array([1.5, 1.5, 1.5]), method="expand", divisions=2)
6 >>> atb.variate(np.array([1.51, 1.51, 1.51]), criteria="right")
7 >>> atb.export("pdf")
```

If your code runs correctly, your files will be stored in the `file_store` directory of your working directory. Inside `file_store`, there will be a folder named after your MIDI file, where you can find all the MIDI files and music sheets created using the above code.

4) Known Issue

For some reason, the `music21` library can generate errors in certain situations. For example, consider the following code:

```
1 >>> import numpy as np
2 >>> from melody import Melody
3 >>> atb = Melody()
4 >>> atb.load("Canon_in_D.mid")
5 >>> atb.fit(np.array([1.5, 1.5, 1.5]), method="expand", divisions=2)
6 >>> atb.variate(np.array([1.51, 1.51, 1.51]), criteria="right")
7 >>> atb.export("midi")
8 >>> atb.export("pdf")
```

If you try to use `atb.fit` or `atb.variate` again after running the above process, you will encounter an error. After conducting several experiments to investigate the cause, we discovered that `music21` shares the MIDI file data across all variables that use the `music21` MIDI reader object. This means that if you attempt to run `atb.variate` multiple times, the result will cause the new variations to expand beyond the intended settings.

We attempted to fix this issue by resetting the object containing the MIDI file data, hoping to make `atb.variate` usable again after generating the MIDI files and sheet music. Unfortunately, due to the shared MIDI file data, there is no way to reset or back up this data to fix the bug.

Additionally, because the MIDI file data is shared, running `atb.fit` after completing the process causes an error since the original MIDI data is modified. The only way to avoid this error is to re-run the entire process, from `atb.load` to `atb.export`. While this workaround may seem unreasonable, especially if you only want to adjust a parameter and rerun a specific function or add a small change and rerun a specific function in your existing code, it is currently the only solution available, as `music21` has not been updated to address this issue.

4.1.2 Web Application

The web application is an application designed for users interested in generating musical variations. It provides an accessible and user-friendly interface that eliminates the need for programming expertise, making it suitable for individuals who wish to create and download musical variations for inspiration.

One of the primary advantages of utilizing this web application is its simplicity. Users can generate musical variations in just two steps: uploading a MIDI file and selecting the option to generate a variation. This streamlined process enables users to apply the chaotic mapping

method or the expanded rhythm method without requiring a deep understanding of the underlying mathematical concepts. Furthermore, the web application includes configurable options that allow users to tailor the generated musical variations to their preferences.

However, a limitation of this web application is its restricted flexibility in the music generation process. Users are constrained to the predefined options available within the application and cannot modify aspects beyond these settings. For advanced customization, users must possess programming knowledge and directly interact with the Python library to make further adjustments.

1) Installation

To install our web application, we recommend using Git, which can be downloaded from <https://git-scm.com/downloads>. First, create a new folder and open a command prompt inside the newly created folder. Then, run the following command:

```
C:\Users\YourFolderLocation> git clone  
→ https://github.com/Ataetano/project_application.git
```

If the command runs successfully, the command prompt should display an output similar to the following:

```
Cloning into 'project_application'...  
remote: Enumerating objects: 1289, done.  
remote: Counting objects: 100% (1289/1289), done.  
remote: Compressing objects: 100% (817/817), done.  
remote: Total 1289 (delta 403), reused 1289 (delta 403), pack-reused 0 (from  
→ 0)  
Receiving objects: 100% (1289/1289), 36.68 MiB | 572.00 KiB/s, done.  
Resolving deltas: 100% (403/403), done.  
Updating files: 100% (1145/1145), done.
```

Next, install Python, which can be downloaded from <https://www.python.org/downloads/>. Once Python is installed, install the required dependencies by running the following commands in the command prompt:

```
pip install -r requirements.txt
```

After successfully installing all the required packages, you are ready to use our web application. The application is located inside the `project_application` folder, specifically within the `web_application` subdirectory. Additionally, the `web_application` folder should contain the following key files: `app.py`, `melody_api.py`, and `lilypond-x-xx-x`.

Lastly, to verify that the installation was successful, open the `run.bat` file located in the `web_application` directory. Upon execution, the web interface should appear as shown in Figure 4.1.

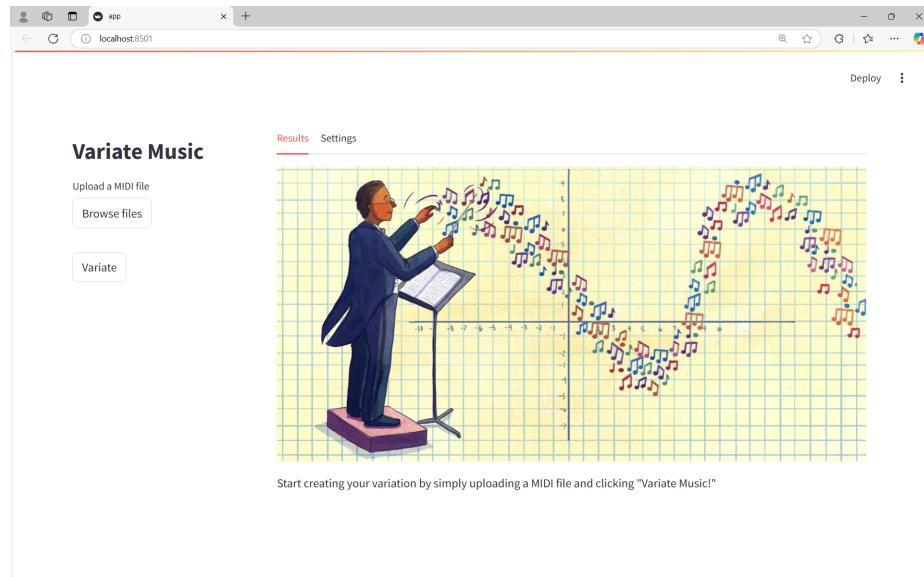
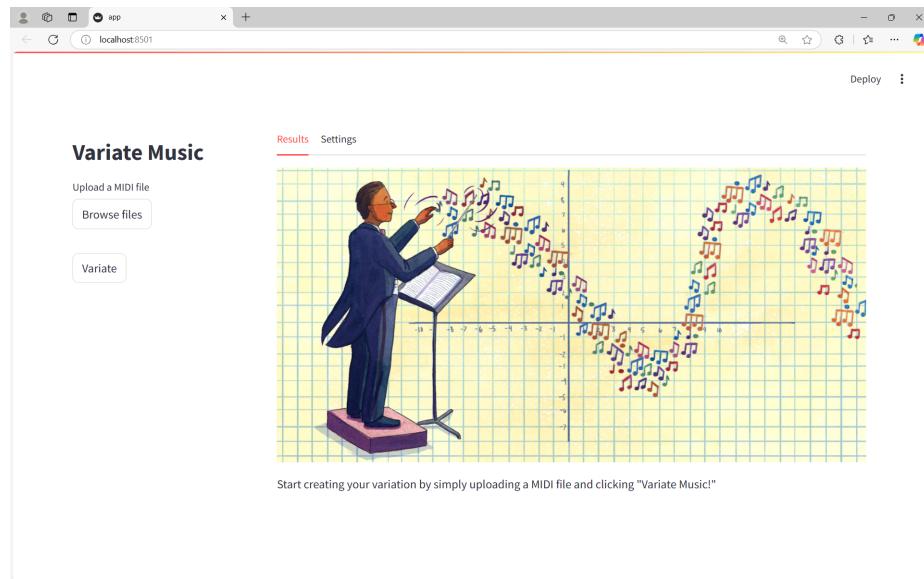


Figure 4.1: Web application interface.

If the web application launches successfully on your browser, the installation has been completed, and you are now ready to use the application. To terminate the web application, simply close the `run.bat` window.

2) Instructions and Example

In this section, we demonstrate the process of generating a musical variation using the web application. To begin, open the `run.bat` file located in the `web_application` directory. This will launch the web application, displaying the following interface:



To generate a musical variation, follow these two steps:

- Step 1:** Click the `Browse files` button and select the MIDI file you want to use for generating a musical variation.

Upload a MIDI file

Browse files

If your file has the .mid extension, the website will display the message: **File saved successfully!**

Generate Music

Upload a MIDI file

Browse files

Uploaded file: CID.mid

File saved successfully!

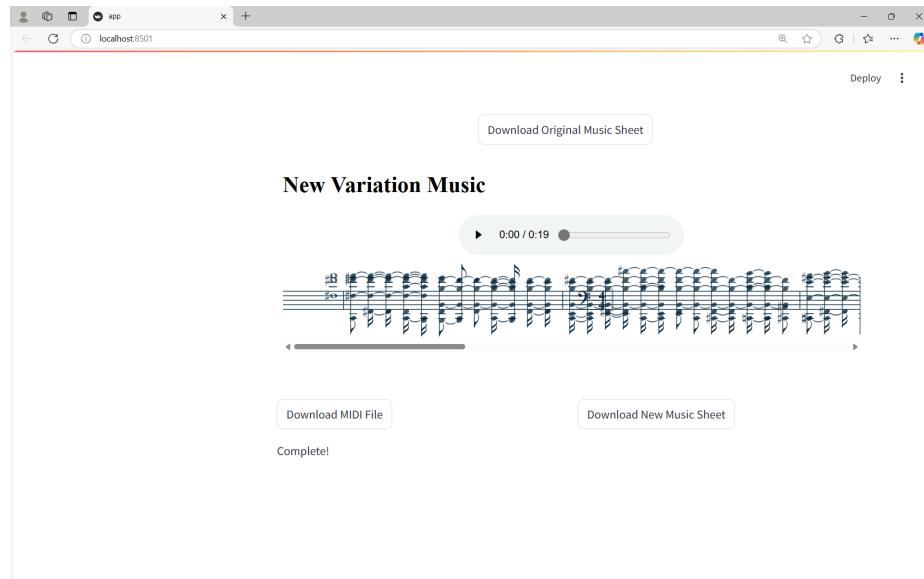
Step 2: Click the **Variate** button to generate a musical variation from your imported MIDI file.

Variate

If the application successfully generates a musical variation, the following result will be displayed:

The screenshot shows a web browser window with the URL `localhost:8501`. The page title is "Variate Music". On the left, there's a "Variate Music" section with a "Upload a MIDI file" input field and a "Browse files" button. Below it, a green box displays the message "File saved successfully!". In the center, there's a "Results" tab selected, showing "Original Music" and "New Variation Music" sections. Each section contains a play button, a progress bar (0:00 / 0:19), and a musical score. At the bottom of each section is a "Download Original Music Sheet" or "Download New Variation Music Sheet" button. The "Settings" tab is also visible at the top of the results section.

You can now listen to your new variation and download the MIDI file and music sheet by scrolling down on the website, where the following section will be displayed:



At this point, you have successfully completed all the steps to generate a musical variation.

To modify the algorithm used for generating a musical variation, click on the **Settings** tab as shown below:

Generate Music

After clicking on the **Settings** tab, you will see the following display:

Variable	Value	Parameter	Value
x	1.01	σ	10.00
y	1.01	ρ	28.00
z	1.01	β	2.67

If you are interested in adjusting these options, please note that some require mathematical knowledge. Below is an explanation of each option:

- **Choose Chaotic System:** Select the dynamical system used for generating the musical variation. Currently, only the Lorenz system is available.
- **Select Main Variable:** Choose the variable from the selected dynamical system.

- **Initial Condition:** Set the initial condition values for the dynamical system.
- **Parameters:** Define the parameter values used in the dynamical system.
- **Check Parameters:** Verify that the selected parameter values ensure the dynamical system exhibits chaotic behavior.
- **Select MIDI Tracks:** (This option appears after generating a musical variation.) Choose the MIDI tracks to generate a new variation. You can select all tracks or specific ones as desired.
- **Choose Method for Generating Musical Variation:** Select the method for generating the variation. Options include the classic method and the expanded rhythm method.

Once you have finalized your settings, return to the **Results** tab to generate a musical variation based on your adjustments.

4.1.3 Implementation

The Python `import` statement is a fundamental construct that facilitates modular programming by allowing the integration of external libraries and modules. This capability enhances code reusability and extends the functionality of a program. In the present implementation, several essential libraries are imported to support the development of a web-based application tailored for processing musical variations through dynamical systems.

In the development of this web application, the following packages are imported:

```

1 import streamlit as st
2 import os
3 from melody_api import Melody
4 from dynamic_system.dynamic_system import *
5 import streamlit.components.v1 as components

```

Each imported module serves a distinct purpose within the application framework:

- Streamlit is utilized for building the interactive user interface.
- The `os` module provides access to operating system functionalities.
- The `melody_api` module handles musical data processing.
- The `dynamic_system.dynamic_system` module is imported to incorporate dynamical system techniques.
- Streamlit's `components.v1` module enables the integration of custom web elements.

For the complete implementation of the web application, please refer to [Source Code](#).

1) Session State Initialization

This section initializes the session state variables used throughout the Streamlit application. The session state is a dictionary-like object that persists across reruns of the app, allowing data to be stored and accessed globally.

1.1) Initialization of Chaotic System Variables

The following code initializes the variables and parameters for the chaotic system:

```
1 if 'x' not in st.session_state:  
2     st.session_state['x'] = 1.01  
3 if 'y' not in st.session_state:  
4     st.session_state['y'] = 1.01  
5 if 'z' not in st.session_state:  
6     st.session_state['z'] = 1.01  
7 if 'sigma' not in st.session_state:  
8     st.session_state['sigma'] = 10.0  
9 if 'rho' not in st.session_state:  
10    st.session_state['rho'] = 28.0  
11 if 'beta' not in st.session_state:  
12    st.session_state['beta'] = 2.67
```

This block:

- Sets default values for the variables x , y , and z , which represent the initial conditions of the chaotic system.
- Initializes the parameters σ , ρ , and β , which control the behavior of the Lorenz system.
- These values are used as starting points for the chaotic system and can be adjusted by the user in the **Settings** tab.

1.2) Initialization of Method and Options

The following code initializes the method and options for melody generation:

```
1 if 'method' not in st.session_state:  
2     st.session_state['method'] = "Classic"  
3 if 'option' not in st.session_state:  
4     st.session_state['option'] = 2  
5 if 'sequence_select' not in st.session_state:  
6     st.session_state['sequence_select'] = 0
```

This block:

- Sets the default method for melody generation to **Classic**.
- Initializes the **option** variable, which determines the expansion factor for the **Expanded** method.
- Sets the default sequence selection to 0, which corresponds to the variable component in the chaotic system.

1.3) Initialization of File and Path Variables

The following code initializes variables related to file uploads and paths:

```
1 if 'uploaded_file' not in st.session_state:  
2     st.session_state['uploaded_file'] = None  
3 if 'save_path' not in st.session_state:  
4     st.session_state['save_path'] = None  
5 if 'original_path' not in st.session_state:  
6     st.session_state['original_path'] = None  
7 if 'new_path' not in st.session_state:  
8     st.session_state['new_path'] = None
```

```

9  if 'midi_new_download_path' not in st.session_state:
10     st.session_state['midi_new_download_path'] = None
11 if 'pdf_original_path' not in st.session_state:
12     st.session_state['pdf_original_path'] = None
13 if 'pdf_new_path' not in st.session_state:
14     st.session_state['pdf_new_path'] = None

```

This block:

- Initializes variables to store the uploaded MIDI file and its paths.
- Sets default values to None for paths related to the original and new MIDI and PDF files.
- These variables are updated when a file is uploaded and processed.

1.4) Initialization of State Flags

The following code initializes flags to track the state of the application:

```

1  if 'generate_clicked' not in st.session_state:
2      st.session_state['generate_clicked'] = False
3  if 'complete_download' not in st.session_state:
4      st.session_state['complete_download'] = False
5  if 'file_exist' not in st.session_state:
6      st.session_state['file_exist'] = 0
7  if 'under_process' not in st.session_state:
8      st.session_state['under_process'] = None

```

This block:

- Initializes `generate_clicked` to track whether the **Variate** button has been clicked.
- Sets `complete_download` to track whether the file download process is complete.
- Initializes `file_exist` to track whether a file has been uploaded.
- Sets `under_process` to track whether the application is currently processing a file.

1.5) Initialization of Byte Streams

The following code initializes variables to store byte streams for file downloads:

```

1  if 'original_pdf_bytes' not in st.session_state:
2      st.session_state['original_pdf_bytes'] = None
3  if 'new_midi_bytes' not in st.session_state:
4      st.session_state['new_midi_bytes'] = None
5  if 'new_pdf_bytes' not in st.session_state:
6      st.session_state['new_pdf_bytes'] = None

```

This block:

- Initializes variables to store byte streams for the original and new PDF and MIDI files.
- These byte streams are used to enable file downloads for users.

1.6) Initialization of Track Lists

The following code initializes variables to store track lists for MIDI processing:

```
1 if 'track_list' not in st.session_state:  
2     st.session_state['track_list'] = None  
3 if 'backup_track_list' not in st.session_state:  
4     st.session_state['backup_track_list'] = None
```

This block:

- Initializes variables to store the current and backup track lists for MIDI files.
- These lists are used to manage and restore tracks during melody generation.

1.7) Initialization of HTML Component

The following code initializes a variable to track the state of the HTML component:

```
1 if 'html_component' not in st.session_state:  
2     st.session_state['html_component'] = None
```

This block:

- Initializes `html_component` to track whether the MIDI player and visualizer have been rendered.
- This variable ensures that the HTML component is only rendered when necessary.

1.8) Initialization of Method and Expanded Indices

The following code initializes indices for method and expanded options:

```
1 if 'method_index' not in st.session_state:  
2     st.session_state['method_index'] = 0  
3 if 'expanded_index' not in st.session_state:  
4     st.session_state['expanded_index'] = 0
```

This block:

- Initializes `method_index` to track the selected method in the dropdown menu.
- Initializes `expanded_index` to track the selected expansion factor in the dropdown menu.

2) Sidebar Interface

This section of the Streamlit application provides a user interface for uploading and processing MIDI files. The interface is implemented using a sidebar layout, which organizes user interaction elements in a structured manner.

2.1) Display Header

The following code snippet displays a title with custom styling:

```
1 st.markdown(  
2     '<h1 style="font-size:30px;">Variate Music</h1>',  
3     unsafe_allow_html=True  
4 )
```

The title “**Variate Music**” is rendered with a font size of **30px**. The parameter `unsafe_allow_html=True` is used to enable raw HTML styling within the Streamlit application.

2.2) File Uploader

A file uploader component is implemented to accept MIDI files:

```
1 uploaded_file = st.file_uploader("Upload a MIDI file", type=["mid", "midi"])
```

This component restricts file uploads to MIDI files with extensions .mid or .midi.

2.3) CSS Styling for File Uploader

Custom CSS is applied to the file uploader to enhance its appearance and functionality:

```
1 css = '''
2 <style>
3     [data-testid='stFileUploader'] {
4         width: max-content;
5     }
6     [data-testid='stFileUploader'] section {
7         padding: 0;
8         float: left;
9     }
10    [data-testid='stFileUploader'] section > input + div {
11        display: none;
12    }
13    [data-testid='stFileUploader'] section + div {
14        display: none;
15    }
16 </style>
17 '''
18 st.markdown(css, unsafe_allow_html=True)
```

The CSS modifications include:

- Setting the width to `max-content`.
- Removing padding and floating the uploader to the left.
- Hiding specific user interface elements of the uploader.

2.4) Handling Uploaded File

The following code block manages the uploaded file:

```
1 if uploaded_file is not None:
2     if st.session_state['file_exist'] == 1:
3         print(f"now_file = {uploaded_file.name}")
4         print(f"exist_file = {st.session_state['uploaded_file'].name}")
5         if uploaded_file.name != st.session_state['uploaded_file'].name:
6             st.session_state['track_list'] = None
7             st.session_state['backup_track_list'] = None
8         st.session_state['uploaded_file'] = uploaded_file
9         st.session_state['file_exist'] = 1
```

This block performs the following tasks:

- Checks if a file has been uploaded.
- Compares the uploaded file with the previously uploaded file.
- Resets the track lists if a new file is uploaded.
- Stores the uploaded file in `st.session_state` for tracking purposes.

2.5) Saving and Processing the MIDI File

The uploaded MIDI file is saved and processed as follows:

```
1 if 'uploaded_file' in st.session_state and st.session_state['uploaded_file']
2     ~ is not None:
3         st.write("Uploaded file:", st.session_state['uploaded_file'].name)
4
5     save_directory = os.path.join("static", "midi_file")
6     os.makedirs(save_directory, exist_ok=True)
7
8     if ' ' in str(st.session_state['uploaded_file'].name):
9         st.warning('Warning: We can\'t save your MIDI file because your MIDI
10             ~ file name has spacing. Please fill the spacing with "_"')
11     else:
12         save_path = os.path.join(save_directory,
13             ~ st.session_state['uploaded_file'].name)
14
15         # Reset file pointer to the beginning
16         st.session_state['uploaded_file'].seek(0)
17
18         # Save the uploaded file locally
19         with open(save_path, "wb") as f:
20             f.write(st.session_state['uploaded_file'].read())
21             st.success("File saved successfully!")
22
23         # Reset file pointer again to read for MIDI processing
24         st.session_state['uploaded_file'].seek(0)
```

This section performs the following operations:

- Displays the name of the uploaded file.
- Creates a directory named static/midi_file to store the MIDI file.
- Issues a warning if the filename contains spaces.
- Saves the uploaded MIDI file locally.
- Resets the file pointer to ensure the file can be read again.

2.6) Loading MIDI Data

The MIDI file is loaded into a Melody object for further processing:

```
1 try:
2     # Load into your Melody object
3     st.session_state['save_path'] = save_path
4     my_melody.load(path=st.session_state['save_path'])
5 except Exception as e:
6     st.error(f"Failed to read MIDI file: {e}")
```

This block attempts to:

- Load the MIDI file into the my_melody object.
- Display an error message if the loading process fails.

2.7) Generate Button

A button is provided to initiate further processing of the uploaded MIDI file:

```
1 generate_button = st.button('Variate')
```

This button, labeled "Variate", triggers additional processing steps for the uploaded MIDI file.

The output generated from the execution of all above implementation is illustrated in Figure 4.2, providing a visual representation of the expected results.

Variate Music

Upload a MIDI file

Browse files

Variate

Figure 4.2: Graphical representation of the interface displaying the Sidebar.

3) Menu and Main Webpage

This section of the Streamlit application defines the main interface, including tabs for displaying results and settings. The interface is organized into two primary tabs: **Results** and **Settings**.

3.1) Tab Layout

The following code initializes the tab layout for the main webpage:

```
1 with right_col:
2     tabs = st.tabs(["Results", "Settings"])
```

This creates two tabs labeled **Results** and **Settings**, allowing users to switch between viewing generated results and adjusting application settings.

3.2) Results Tab

The results tab handles the display of generated music variations and user interactions.

3.2.1) Initial State

When the application starts, the following code checks if the `generate_button` has been clicked:

```
1 if generate_button:
2     st.session_state['generate_clicked'] = True
```

If the button has not been clicked, an introductory image and message are displayed:

```

1 if st.session_state['generate_clicked'] == False:
2     st.image("intro.jpg")
3     st.write("Start creating your variation by simply uploading a MIDI file
4         ↵ and clicking \"Variate Music!\"")

```

This provides users with instructions on how to use the application.

3.2.2) File Processing

If a MIDI file has been uploaded, the following code processes it:

```

1 if 'uploaded_file' in st.session_state and st.session_state['uploaded_file']
2     ↵ is not None:
3         if generate_button == True:
4             st.session_state['complete_download'] = False
5             my_melody.set_dynamic_sequence(sequence=st.session_state['sequence_se_'
6                 ↵ lect'])
7             key_list =
8                 ↵ my_melody.fit(original_initial_condition=[st.session_state['x'],
9                     ↵ st.session_state['y'], st.session_state['z']],
10                     method=st.session_state['method'],
11                     ↵ divisions=st.session_state['option'],
12                     dynamic=le, d=st.session_state['sigma'],
13                     ↵ r=st.session_state['rho'], b=st.session_state['beta'])
14             if st.session_state['track_list'] == None:
15                 st.session_state['track_list'] = [key for key in key_list]
16                 st.session_state['backup_track_list'] = [key for key in key_list]
17             my_melody.variate(new_initial_condition=[1.5, 1.5, 1.5],
18                 ↵ track=st.session_state['track_list'])

```

This block:

- Sets the dynamic sequence for the melody based on user-selected variables.
- Fits the melody to the selected chaotic system parameters.
- Generates a new variation of the melody using the provided initial conditions.

3.2.3) Exporting Results

The generated MIDI and PDF files are exported and made available for download:

```

1 original_path, new_path, midi_new_download_path = my_melody.export("midi")
2 st.session_state['original_path'] = original_path
3 st.session_state['new_path'] = new_path
4 st.session_state['midi_new_download_path'] = midi_new_download_path
5 with open(st.session_state['midi_new_download_path'], "rb") as file:
6     st.session_state['new_midi_bytes'] = file.read()
7
8 pdf_original_path, pdf_new_path = my_melody.export("pdf")
9 st.session_state['pdf_original_path'] = pdf_original_path
10 with open(st.session_state['pdf_original_path'], "rb") as file:
11     st.session_state['original_pdf_bytes'] = file.read()
12
13 st.session_state['pdf_new_path'] = pdf_new_path
14 with open(st.session_state['pdf_new_path'], "rb") as file:
15     st.session_state['new_pdf_bytes'] = file.read()

```

This section:

- Exports the original and new MIDI files.
- Exports the original and new PDF files (sheet music).
- Prepares the files for download by reading them into byte streams.

3.2.4) MIDI Visualizer

The following code embeds MIDI players and visualizers for the original and new variations:

```

1 components.html(
2 f"""
3 <h1 style="font-size:30px;">Original Music</h1>
4 <center>
5 <midi-player
6   src="{st.session_state['original_path']}"
7   sound-font visualizer="#myStaffVisualizer1">
8 </midi-player>
9
10 <midi-visualizer type="staff" id="myStaffVisualizer1"
11   src="{st.session_state['original_path']}"/>
12 </midi-visualizer>
13 </center>
14 <script src="https://cdn.jsdelivr.net/combine/npm/tone@14.7.58,npm/@magenta/m
15   usic@1.23.1/es6/core.js,npm/focus-visible@5,npm/html-midi-player@1.5.0"><
16   /script>
17 """
18 , height=300)
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
235
236
237
237
238
239
239
240
241
242
242
243
243
244
244
245
245
246
246
247
247
248
248
249
249
250
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
138
```

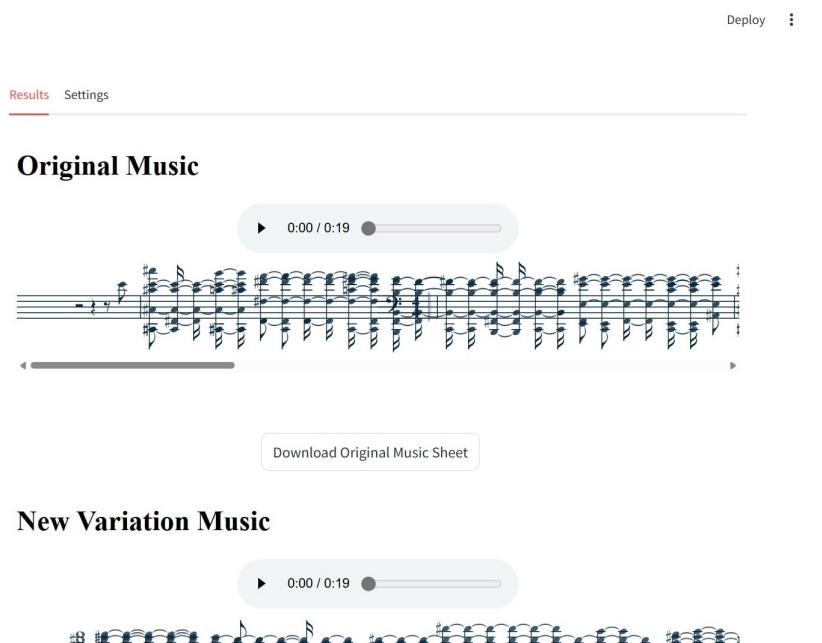
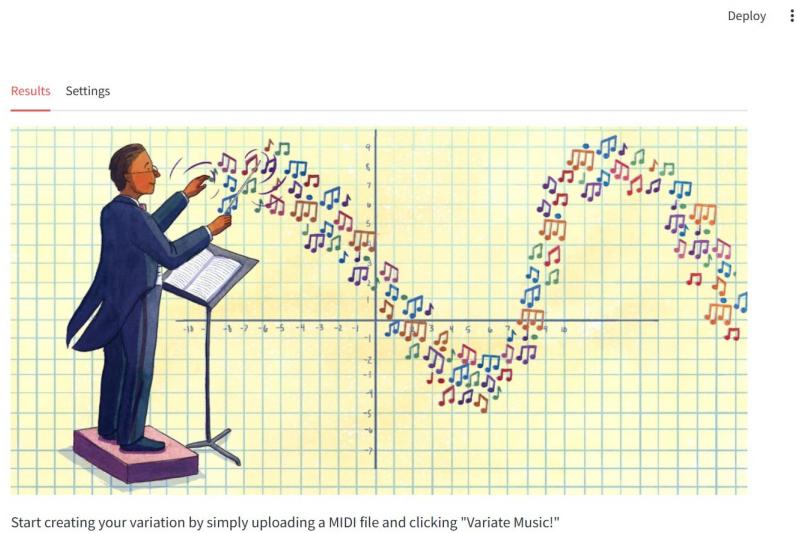


Figure 4.3: Graphical representation of the interface displaying the results webpage.

3) Settings Tab

The **Settings** tab allows users to configure parameters for the chaotic systems used in melody generation.

3.3.1) Chaotic System Selection

Users can select a chaotic system from a dropdown menu:

```

1 equation = st.selectbox(
2     "Choose Chaotic Systems",
3     ("Lorenz system", "Chaotic system"),
4     index=0,
5     placeholder="Choose Equation"
6 )

```

This dropdown provides options for selecting the chaotic system to be used.

3.3.2) Variable and Parameter Configuration

For the Lorenz system, users can configure variables and parameters:

```
1 if equation == "Lorenz system":
2     variable_Lorenz = st.selectbox(
3         "Select variable to display",
4         options=("x", "y", "z"),
5         index=0
6     )
7     st.session_state['x'] = st.slider("x", min_value=-5.0, max_value=5.0,
8         value=1.01, step=0.01)
9     st.session_state['y'] = st.slider("y", min_value=-5.0, max_value=5.0,
10        value=1.01, step=0.01)
11    st.session_state['z'] = st.slider("z", min_value=-5.0, max_value=5.0,
12        value=1.01, step=0.01)
13    st.session_state['sigma'] = st.slider(r"\sigma", min_value=0.0,
14        max_value=100.0, value=10.0, step=0.01)
15    st.session_state['rho'] = st.slider(r"\rho", min_value=0.0,
16        max_value=100.0, value=28.0, step=0.01)
17    st.session_state['beta'] = st.slider(r"\beta", min_value=0.0,
18        max_value=100.0, value=2.67, step=0.01)
```

This block:

- Allows users to select a variable (x, y, or z) for visualization.
- Provides sliders for adjusting the initial conditions and parameters of the Lorenz system.

3.3.3) Parameter Validation

Users can validate the chaotic parameters:

```
1 if st.button('Check Parameters'):
2     my_melody.load(path=st.session_state['save_path'])
3     my_melody.set_dynamic_sequence(sequence=st.session_state['sequence_select'],
4         ''))
5     my_melody.fit(initial_condition=[st.session_state['x'], st.session_state['y'],
6         st.session_state['z']], dynamic=le, d=st.session_state['sigma'],
7         r=st.session_state['rho'], b=st.session_state['beta'])
8     checker = my_melody.is_chaotic()
9     if str(checker) == 'True':
10         st.write(f"Chaotic parameter")
11     else:
12         st.warning('Parameter not chaotic')
```

This section:

- Loads the MIDI file and applies the selected parameters.
- Checks if the parameters produce chaotic behavior and displays the result.

3.3.4) Additional Options

Users can select additional options for melody generation:

```
1 st.session_state['method'] = st.selectbox(
2     "Additional Options",
3     ("Classic", "Expanded"),
4     index=st.session_state['method_index'],
5     placeholder="Choose Equation",
6 )
```

This dropdown allows users to choose between **Classic** and **Expanded** methods for generating variations.

3.3.5) Expanded Method Configuration

If the **Expanded** method is selected, users can configure the expansion factor:

```

1 if st.session_state['method'] == "Expanded":
2     st.session_state['option'] = st.selectbox(
3         "Expanded Number (Example: If you choose 4, our algorithm will expand
4             ↵ 1 note into 4 notes in your MIDI file)",
5         (2, 4, 8, 16),
6         index=st.session_state['expanded_index'],
7     )

```

This block:

- Allows users to specify how many notes each original note should be expanded into.
- Updates the session state with the selected expansion factor.

The output generated from the execution of all above implementation is illustrated in Figure 4.4, providing a visual representation of the expected results.

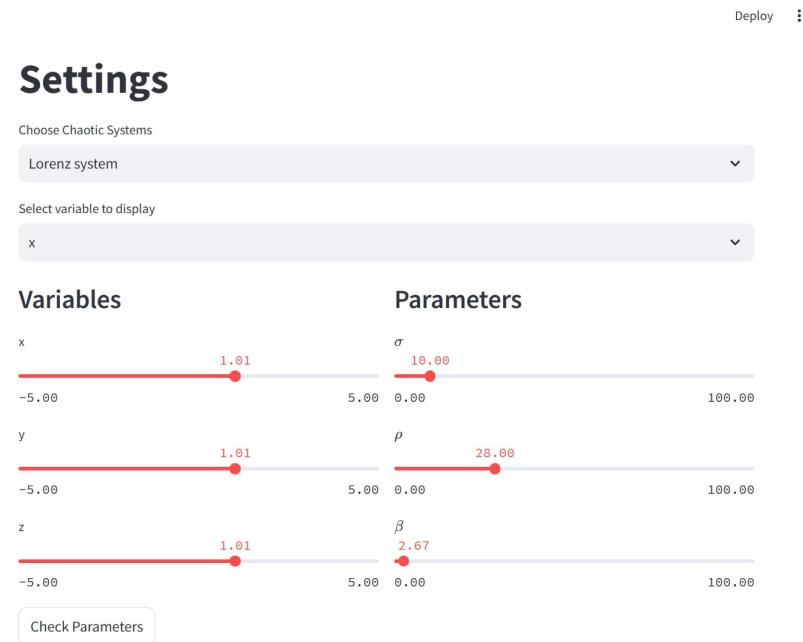


Figure 4.4: Graphical representation of the interface displaying the settings webpage.

4.2 Conclusion

The proposed method successfully achieves all three objectives by developing an alternative algorithm that requires fewer resources for music generation through a chaotic dynamical system. This approach enables the creation of new compositions based on existing musical elements while also providing composers with limited resources access to a tool that serves as inspiration for further songwriting. As a result, this method introduces a novel approach to music composition that fosters innovation and diversity in musical expression.

The Python library is designed for students and researchers interested in this project, aiming to facilitate study and further development. Its key advantage lies in its flexibility to

configure functions for generating musical variations, such as adjusting parameters or adding new functions to enhance performance. However, a drawback is that users need a basic understanding of Python programming to utilize it effectively.

The web application is designed to allow users to easily generate new musical variations from MIDI files through an automated process, without requiring programming skills. However, while it provides options for adjusting various parameters of music generation, users cannot fully control the process freely. If they wish to make deeper customizations, they will need to use the Python library developed in this project to write code and define musical patterns according to their preferences.

Finally, the proposed method, based on the dynamics of chaotic systems, introduces a new approach to music composition that overcomes AI limitations and reduces computational demands. It sparks creativity and helps alleviate composer burnout. By practically applying chaos, this method enables composers to explore new possibilities, break free from conventional constraints, and potentially transform both the composition process and musical experience. It envisions a future where unpredictability and diversity become the essence of musical creativity.

References

- [1] A. Bonnici et al. *Music and AI*. Frontiers Media SA, 2021. 170 pp. ISBN: 978-2-88966-602-7.
- [2] Mubert Inc. *Mubert - Human and AI Music Generator for Your Video Content, Podcasts and Apps*. 2024. URL: <https://mubert.com/> (visited on 04/24/2024).
- [3] Musicfy Inc. *Musicfy - Change Your Voice with AI*. 2023. URL: <https://musicfy.lol/> (visited on 04/27/2024).
- [4] Soundraw Inc. *Soundraw - The Music Tool for Creators and Artists*. 2020. URL: <https://soundraw.io> (visited on 04/24/2024).
- [5] Boomy Corporation. *Boomy - Unleash Your Creativity. Make Music with Boomy AI*. 2023. URL: <https://boomy.com/> (visited on 04/24/2024).
- [6] Aiva Technologies SARL. *AIVA - Your Personal AI Music Generation Assistant*. 2024. URL: <https://www.aiva.ai> (visited on 04/24/2024).
- [7] A. Mushtaq. *A SOLUTION FOR ORDINARY DIFFERENTIAL EQUATION: SOLVING TECHNIQUES AND APPLICATIONS*. Horizon Books (A Division of Ignited Minds Edutech P Ltd), 2015. ISBN: 9789384044794. URL: <https://books.google.co.th/books?id=ChpKDwAAQBAJ>.
- [8] J. Adamy. *Nonlinear Systems and Controls*. Springer Berlin Heidelberg, Imprint: Springer, 2024. ISBN: 9783662686904. URL: <https://books.google.co.th/books?id=eDz-EAAAQBAJ>.
- [9] R. L. Burden and J. D. Faires. *Numerical Analysis*. 9th ed. Boston, MA: Cengage Learning, 2010.
- [10] H. Anton and C. Rorres. *Elementary Linear Algebra: Applications Version*. 11th. Hoboken, NJ: John Wiley and Sons, 2013. ISBN: 978-1-118-43441-3.
- [11] Jitka Khůlová. “Stability and chaos in nonlinear dynamical systems”. Master’s thesis. Brno, Czech Republic: Brno University of Technology, Faculty of Mechanical Engineering, Institute of Mathematics, 2018, p. 74.
- [12] E. Tlelo-Cuautle et al. *Optimization of Integer/Fractional Order Chaotic Systems by Metaheuristics and their Electronic Realization*. CRC Press, 2021. ISBN: 9781000346657. URL: <https://books.google.co.th/books?id=jq4hEAAAQBAJ>.
- [13] D. S. Dabby. “Musical variations from a chaotic mapping”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 6.2 (1996), pp. 95–107. ISSN: 1054-1500. doi: [10.1063/1.166171](https://doi.org/10.1063/1.166171).
- [14] M. Hinson. *12 Variations on Ah, Vous Dirai-Je, Maman, K.* 265. ALFRED PUBN, 1987. 16 pp. ISBN: 978-0-7390-2032-6.

Appendix

Source Code

Python Library

The Python library source code for this project includes the main class, instance variables, a method for generating musical variations from a chaotic mapping, and a method for generating musical variations from a expanded rhythm:

```
1 import music21
2 from music21 import stream, midi, scale, converter, instrument
3 from datetime import datetime as dt
4 import numpy as np
5 import subprocess
6 import os
7 import pathlib
8 import matplotlib.pyplot as plt
9
10 class Melody:
11
12     def __init__(self):
13         self.tracks = []
14         self.maps = []
15         self.dynamic = self.lorenz_system
16
17         # Main Instance Attributes
18         self.original_midi_data = None
19         self.new_midi_data = None
20
21         # Additional Instance Attributes
22         self.newtracks = []
23         self.main_stream = None
24         self.new_stream = None
25         self.duration = []
26         self.new_duration = []
27         self.original_initial_condition = None
28         self.new_initial_condition = None
29         self.dynamic_sequence = 0
30         self.notetrack = []
31         self.tracks_index = []
32         self.dynamic_parameter = []
33
34         # Support Instance Attributes
35         self.method = None
36         self.divisions = None
37         self.backtrack_expand_index = []
38
39         # Path
```

```

40     self.lilypond_path =
41         str(pathlib.Path("__file__").parent.resolve()) +
42         "\\\lilypond-2.24.4\\bin\\lilypond.exe"
43
44     # etc.
45     self.filename = None
46
47 ######
48
49 def load(self, path):
50     midi_file = midi.MidiFile()
51     midi_file.open(path)
52     midi_file.read()
53     midi_file.close()
54
55     self.original_midi_data =
56         midi.translate.midiFileToStream(midi_file)
57     self.new_midi_data = midi.translate.midiFileToStream(midi_file)
58
59     self.main_stream = midi.translate.midiFileToStream(midi_file)
60     self.new_stream = midi.translate.midiFileToStream(midi_file)
61     self.filename = str(pathlib.Path(path).stem)
62
63 def fit(self, original_initial_condition, method="classic",
64        divisions=None):
65     self.original_initial_condition = original_initial_condition
66     self.method = method
67     self.divisions = divisions
68
69     if self.method == "classic":
70         for index in range(len(self.main_stream)):
71             instruments = self.main_stream[index][0].getElementsBy_
72                 Class(instrument.Instrument)
73             if instruments:
74                 instrument_name =
75                     str(instruments[0].instrumentName) + "_" +
76                     str(index)
77             else:
78                 instrument_name = "track_" + str(index)
79             music_list = []
80             duration_list = []
81             for part in self.main_stream[index]:
82                 for n in part.flat.notesAndRests:
83                     dur = n.duration.quarterLength
84                     if n.isRest:
85                         note_num = "<REST>"
86                         music_list.append(note_num)
87                         duration_list.append(dur)
88                     else:
89

```

```

84         if len(n.pitches) == 1:
85             note_num = n.pitches[0].midi
86             music_list.append(note_num)
87             duration_list.append(dur)
88         else:
89             note_num = sorted(set([p.midi for p in
90             ↪ n.pitches]))
91             music_list.append(list(note_num))
92             duration_list.append(dur)
93     string_music_list = [element for element in music_list]
94     for i in range(len(music_list)):
95         try:
96             string_music_list[i] =
97             ↪ self.note_converter(music_list[i])
98         except TypeError:
99             for j in range(len(music_list[i])):
100                 string_music_list[i][j] =
101                 ↪ self.note_converter(music_list[i][j])
102
103     time, solution = self.rk4(time=[0,
104     ↪ 0.01*(len(music_list) - 1)], f=self.dynamic,
105     ↪ ini=self.original_initial_condition,
106     ↪ **self.dynamic_parameter)
107     main_sequence = solution[:, self.dynamic_sequence]
108     print(main_sequence[0:100:5])
109     #self.original_time = time
110     # self.original_trajectory = main_sequence
111
112     self.notetrack.update({instrument_name:string_music_li
113     ↪ st})
114     self.duration.update({instrument_name:duration_list})
115     self.tracks.update({instrument_name:main_sequence})
116     self.tracks_index.update({instrument_name:index})
117     for key in self.tracks.keys():
118         dummy_list = []
119         for i in range(len(self.notetrack[key])):
120             dummy_list.append([self.tracks[key][i],
121             ↪ self.notetrack[key][i]])
122             self.maps.update({key:dummy_list})
123     elif self.method == "expand":
124         for index in range(len(self.main_stream)):
125             instruments = self.main_stream[index][0].getElementsBy
126             ↪ Class(instrument.Instrument)
127             if instruments:
128                 instrument_name =
129                 ↪ str(instruments[0].instrumentName) + "_" +
130                 ↪ str(index)
131             else:
132                 instrument_name = "track_" + str(index)
133             duration_list = []
134             for part in self.main_stream[index]:

```

```

124     for n in part.flat.notesAndRests:
125         dur = n.duration.quarterLength
126         if n.isRest:
127             duration_list.append(dur)
128         else:
129             if len(n.pitches) == 1:
130                 duration_list.append(dur)
131             else:
132                 duration_list.append(dur)
133         self.duration.update({instrument_name:duration_list})
134     for index in range(len(self.main_stream)):
135         self.expand_note_durations(s=self.main_stream[index],
136                                     divisions=self.divisions)
137         instruments = self.main_stream[index][0].getElementsBy_
138                                     Class(instrument.Instrument)
139         if instruments:
140             instrument_name =
141                 str(instruments[0].instrumentName) + "_" +
142                 str(index)
143         else:
144             instrument_name = "track_" + str(index)
145         music_list = []
146         #duration_list = []
147         for part in self.main_stream[index]:
148             for n in part.flat.notesAndRests:
149                 dur = n.duration.quarterLength
150                 if n.isRest:
151                     note_num = "<REST>"
152                     music_list.append(note_num)
153                     #duration_list.append(dur)
154                 else:
155                     if len(n.pitches) == 1:
156                         note_num = n.pitches[0].midi
157                         music_list.append(note_num)
158                         #duration_list.append(dur)
159                     else:
160                         note_num = sorted(set([p.midi for p in
161                                     n.pitches]))
162                         music_list.append(list(note_num))
163                         #duration_list.append(dur)
164         string_music_list = [element for element in music_list]
165         for i in range(len(music_list)):
166             try:
167                 string_music_list[i] =
168                     self.note_converter(music_list[i])
169             except TypeError:
170                 for j in range(len(music_list[i])):
171                     string_music_list[i][j] =
172                         self.note_converter(music_list[i][j])
173         self.backtrack_expand_index = [index for index in
174                                     range(0, len(music_list) - 1, self.divisions)]

```

```

167     time, solution = self.rk4(time=[0,
168         ← 0.01*(len(music_list) - 1)], f=self.dynamic,
169         ← ini=self.original_initial_condition,
170         ← **self.dynamic_parameter)
171     main_sequence = solution[:, self.dynamic_sequence]
172     #self.original_time = time
173     # self.original_trajectory = main_sequence
174
175     self.notetrack.update({instrument_name:string_music_li,
176         ← st})
177     self.tracks.update({instrument_name:main_sequence})
178     self.tracks_index.update({instrument_name:index})
179     for key in self.tracks.keys():
180         dummy_list = []
181         for i in range(len(self.notetrack[key])):
182             dummy_list.append([self.tracks[key][i],
183                 ← self.notetrack[key][i]])
184         self.maps.update({key:dummy_list})
185     else:
186         raise Exception("Out of Option!")
187     #return self.maps
188     # return self.tracks.keys()
189     #return print(self.tracks.keys())
190
191     def variate(self, new_initial_condition, track=None,
192         ← criteria="right", add_note=0):
193         self.new_initial_condition = new_initial_condition
194         if track == None:
195             track = list(self.tracks.keys())
196         if add_note != 0:
197             for key in track:
198                 for each_note in range(add_note):
199                     # Access parts from the stream
200                     parts = self.new_stream.getElementsByClass('Part')
201                     target_part = parts[self.tracks_index[key]]
202
203                     # Determine the last measure in the part
204                     if target_part.measure(-1):
205                         last_measure = target_part.measure(-1) # Get
206                         ← the last measure
207                     else:
208                         # If no measures exist, create the first one
209                         last_measure = music21.stream.Measure()
210                         target_part.append(last_measure)
211
212                     # Create the new note
213                     new_note = music21.note.Note('C4')
214                     duration_len = len(self.duration[key])
215                     new_note.quarterLength =
216                         ← self.duration[key][each_note % duration_len]
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309

```

```

210         # Append the new note to the last measure
211         last_measure.append(new_note)
212         #print(target_part[-1].show("text"))
213         #print("-"*100)
214
215     if self.method == "expand":
216         for index in range(len(self.new_stream)):
217             self.expand_note_durations(s=self.new_stream[index],
218                                         divisions=self.divisions)
219             instruments = self.new_stream[index][0].getElementsByC_
220                                         lass(instrument.Instrument)
221             if instruments:
222                 instrument_name =
223                     ↵ str(instruments[0].instrumentName) + "_" +
224                     ↵ str(index)
225             else:
226                 instrument_name = "track_" + str(index)
227             duration_list = []
228             for part in self.new_stream[index]:
229                 for n in part.flat.notesAndRests:
230                     dur = n.duration.quarterLength
231                     duration_list.append(dur)
232             self.new_duration.update({instrument_name:duration_list,
233                                         ↵ t})
234             #self.duration = self.new_duration
235
236         for key in track:
237             sorted_tracks = sorted(self.maps[key], key=lambda x: x[0])
238             if self.method == "classic":
239                 dummy_list = [None]*(len(self.tracks[key]) + add_note)
240                 time, solution = self.rk4(time=[0,
241                                             ↵ 0.01*(len(dummy_list) - 1)], f=self.dynamic,
242                                             ↵ ini=self.new_initial_condition,
243                                             ↵ **self.dynamic_parameter)
244                 main_sequence = solution[:, self.dynamic_sequence]
245             elif self.method == "expand":
246                 dummy_list = [None]*(len(self.tracks[key]) +
247                                         ↵ add_note)*self.divisions
248                 time, solution = self.rk4(time=[0,
249                                             ↵ 0.01*(len(dummy_list) - 1)], f=self.dynamic,
250                                             ↵ ini=self.new_initial_condition,
251                                             ↵ **self.dynamic_parameter)
252                 main_sequence = solution[:, self.dynamic_sequence]
253                 # self.new_time = time
254                 # self.new_trajectory = main_sequence
255             else:
256                 raise Exception("Out of Option!")
257
258         if criteria == "right":
259             for i in range(len(main_sequence)) if len(main_sequence)
260                                         ↵ == len(dummy_list) else len(main_sequence) - 1):

```

```

248     for j in range(len(sorted_tracks)):
249         if main_sequence[i] <= sorted_tracks[j][0]:
250             dummy_list[i] = sorted_tracks[j][1]
251             break
252         elif main_sequence[i] > sorted_tracks[-1][0]:
253             dummy_list[i] = sorted_tracks[-1][1]
254             break
255     elif criteria == "left":
256         for i in range(len(main_sequence) if len(main_sequence)
257                         == len(dummy_list) else len(main_sequence) - 1):
258             for j in range(len(sorted_tracks)):
259                 if main_sequence[i] <= sorted_tracks[j][0]:
260                     dummy_list[i] = sorted_tracks[j - 1][1]
261                     break
262                 elif main_sequence[i] > sorted_tracks[-1][0]:
263                     dummy_list[i] = sorted_tracks[-1][1]
264                     break
265                 elif (main_sequence[i] <= sorted_tracks[0][0])
266                         and (j - 1 < 0):
267                     main_sequence[i] = sorted_tracks[0][0]
268                     break
269     else:
270         raise Exception("Out of Option!")
271
272         self.newtracks.update({key:dummy_list})
273
274 #return self.newtracks
275
276
277 def export(self, format_type):
278     exist_track = [track for track in self.newtracks.keys()]
279     exist_new_stream = self.new_midi_data
280     if self.method == "classic":
281         for key in self.tracks.keys():
282             if key in exist_track:
283                 index = 0
284                 for element in exist_new_stream[self.tracks_index[ ]
285                               ↵ key]].recurse():
286                     if isinstance(element, (music21.note.Note,
287                                     ↵ music21.chord.Chord, music21.note.Rest)):
288                         replacement = self.newtracks[key][index]
289                         duration_value = self.duration[key][index]
290                         if replacement == '<REST>':
291                             new_element = music21.note.Rest()
292                         elif isinstance(replacement, list):
293                             new_element =
294                                 ↵ music21.chord.Chord(replacement)
295                         else:
296                             new_element =
297                                 ↵ music21.note.Note(replacement)
298                         new_element.duration = music21.duration.Du
299                             ↵ ration(duration_value)

```

```

291             element.activeSite.replace(element,
292                 ↵ new_element)
293             #print(f"{element} --> {replacement}")
294             if index < len(self.newtracks[key]) - 1:
295                 index += 1
296             else:
297                 break
298         elif self.method == "expand":
299             for key in self.tracks.keys():
300                 if key in exist_track:
301                     index = 0
302                     for element in exist_new_stream[self.tracks_index[,
303                         ↵ key]].recurse():
304                         if isinstance(element, (music21.note.Note,
305                             ↵ music21.chord.Chord, music21.note.Rest)):
306                             replacement = self.newtracks[key][self.bac,
307                                 ↵ ktrack_expand_index[index]]
308                             duration_value = self.duration[key][index]
309                             if replacement == '<REST>':
310                                 new_element = music21.note.Rest()
311                             elif isinstance(replacement, list):
312                                 new_element =
313                                     ↵ music21.chord.Chord(replacement)
314                             else:
315                                 new_element =
316                                     ↵ music21.note.Note(replacement)
317                                 new_element.duration = music21.duration.Du,
318                                     ↵ ration(duration_value)
319                                 element.activeSite.replace(element,
320                                     ↵ new_element)
321                                 #print(f"{element} --> {replacement}")
322                                 if index < len(self.backtrack_expand_index)
323                                     ↵ - 1:
324                                         index += 1
325                                     else:
326                                         break
327                                         #print("-"*50)
328                                         #print(self.new_stream.show("text"))
329                                         current_path = pathlib.Path("__file__").parent.resolve()
330                                         date = str(dt.now().isoformat())[:-7]
331                                         filepath = str(current_path) + "\\\" + "file_store" + "\\\" +
332                                             ↵ self.filename #+ date.replace(":", "-")
333                                         isExist = os.path.exists(filepath)
334                                         if not isExist:
335                                             os.makedirs(filepath)
336                                         original = self.original_midi_data
337                                         new = self.new_midi_data
338                                         if format_type == "midi":
339                                             # original.metadata = music21.metadata.Metadata()
340                                             original.write('midi', filepath + "\\\" + self.filename +
341                                                 ↵ '_original.mid')

```

```

331     # new.metadata = music21.metadata.Metadata()
332     new.write('midi', filepath + "\\" + self.filename +
333             '_new.mid')
334     elif format_type == "pdf":
335         us = music21.environment.UserSettings()
336         us['lilypondPath'] = self.lilypond_path
337
338         original.metadata = music21.metadata.Metadata()
339         original.metadata.title = self.filename.replace("_", " ")
340             #+ " Original"
341         conv = music21.converter.subConverters.ConverterLilypond()
342         original_filepath = filepath + "\\" + self.filename +
343             '_original'
344         conv.write(original, fmt='lilypond', fp=original_filepath,
345             subformats = ['pdf'])
346
347         new.metadata = music21.metadata.Metadata()
348         new.metadata.title = "Our Variation"
349             #self.filename.replace("_", " ") + " New"
350         new_conv =
351             # music21.converter.subConverters.ConverterLilypond()
352         new_filepath = filepath + "\\" + self.filename + '_new'
353         conv.write(new, fmt='lilypond', fp=new_filepath, subformats
354             = ['pdf'])
355
356 ######
357 # Additional Method
358 #####
359
360 def show_all_track(self):
361     return self.tracks.keys()
362
363 def set_dynamic(self, dynamic, *args, **kwargs):
364     self.dynamic = dynamic
365     self.dynamic_parameter.update(*args, **kwargs)
366
367 def set_dynamic_sequence(self, sequence):
368     self.dynamic_sequence = sequence - 1
369
370 # def plot_trajectory(self):
371 #     plt.figure(figsize=(12,6), dpi=240)
372 #     plt.xlabel("time")
373 #     plt.ylabel("trajectory value at time")
374 #     plt.plot(self.original_time, self.original_trajectory,
375             # label="Original Trajectory")
376 #     plt.plot(self.new_time, self.new_trajectory, label="New
377             # Trajectory")
378 #     plt.grid()
379 #     plt.legend()
380 #     plt.show()

```

```

373
374 ######
375 # Support Method
376 #####
377
378 def lorenz_system(self, t, x, sigma=10, rho=28, beta=8/3):
379     """
380         Computes the derivatives of the Lorenz system at a given time
381         → t.
382     Parameters:
383         t (int or float): Time variable (not explicitly used in
384             ← computation).
385         x (array_like): State vector at time t, given as a list or
386             ← NumPy array [x_1, x_2, x_3].
387         sigma (float): Lorenz system parameter (must be greater than
388             ← 0).
389         rho (float): Lorenz system parameter (must be greater than 0).
390         beta (float): Lorenz system parameter (must be greater than 0).
391
392     Returns:
393         numpy.ndarray: The time derivatives [dx/dt, dy/dt, dz/dt] as a
394             ← NumPy array.
395     """
396
397     x_1, x_2, x_3 = x
398     xdot = sigma * (x_2 - x_1)
399     ydot = x_1 * (rho - x_3) - x_2
400     zdot = x_1 * x_2 - beta * x_3
401     return np.array([xdot, ydot, zdot])
402
403
404 def rk4(self, time, ini, f, h=0.01, *args, **kwargs):
405     t = np.arange(time[0], time[1]+h, h)
406     row = len(t)
407     try:
408         var = len(ini)
409     except TypeError:
410         var = 1
411     x = np.zeros((row, var))
412     x[0, :] = ini
413     for j in range(0, len(t)-1):
414         k1 = f(t[j], x[j], *args, **kwargs)
415         k2 = f(t[j] + 0.5*h, x[j] + 0.5*h*k1, *args, **kwargs)
416         k3 = f(t[j] + 0.5*h, x[j] + 0.5*h*k2, *args, **kwargs)
417         k4 = f(t[j] + h, x[j] + h*k3, *args, **kwargs)
418         x[j+1, :] = x[j] + (h/6)*(k1 + 2*k2 + 2*k3 + k4)
419     return t, x
420
421
422 def is_chaotic(self):
423     T = [0, 100]
424
425     # Initial state and parameters
426     initial_state = np.array([1.0, 1.0, 1.0])

```

```

419
420     # Integrate system
421     #sol = solve_ivp(lorenz, t_span, initial_state, t_eval=t_eval,
422     #    ↪ rtol=1e-9, atol=1e-9)
423     time, sol = self.rk4(time=T, ini=initial_state, f=self.dynamic,
424     #    ↪ **self.dynamic_parameter)
425
426
427     # Perturb the initial state slightly
428     perturbed_state = np.array([1.00001, 1.0, 1.0])
429
430
431     #perturbed_sol = solve_ivp(lorenz, t_span, perturbed_state,
432     #    ↪ t_eval=t_eval, rtol=1e-9, atol=1e-9)
433     time, perturbed_sol = self.rk4(time=T, ini=perturbed_state,
434     #    ↪ f=self.dynamic, **self.dynamic_parameter)
435
436
437     # Compute divergence between trajectories
438     divergence = np.linalg.norm(sol - perturbed_sol, axis=1)
439
440
441     # Estimate Lyapunov exponent
442     divergence_rate = np.log(divergence + 1e-10) # Avoid log(0)
443     lyapunov_exponent = np.mean(divergence_rate[time > 10]) #
444     #    ↪ Ignore transient phase
445
446
447     # Chaos detected if Lyapunov exponent is positive
448     return lyapunov_exponent > 0
449
450
451     def note_converter(self, midi_note):
452         """
453             This function convert single midi note value to readable
454             ↪ musical note.
455             Important parameter:
456             midi_note: midi note value;
457                 60 ----> C4
458         """
459
460         if midi_note == '<REST>':
461             return '<REST>'
462         else:
463             if 0 <= midi_note <= 127:
464                 pitches = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G',
465                 #    ↪ 'G#', 'A', 'A#', 'B']
466
467                 octave = (midi_note // 12) - 1
468                 pitch_index = midi_note % 12
469
470                 pitch = pitches[pitch_index]
471                 return f'{pitch}{octave}'
472             else:
473                 return "Invalid MIDI note value. Must be between 0 and
474                 #    ↪ 127."
475
476
477     def expand_note_durations(self, s, divisions):

```

```

462     for measure in s.getElementsByClass(stream.Measure):
463         for element in measure:
464             if isinstance(element, music21.note.Note):
465                 duration = element.duration.quarterLength
466                 expanded_duration = (1 / divisions) * duration
467                 offset = element.offset
468                 measure.remove(element)
469                 for i in range(divisions):
470                     new_note = music21.note.Note(element.pitch)
471                     new_note.duration.quarterLength =
472                         ↵ expanded_duration
473                     measure.insert(offset + i * expanded_duration,
474                         ↵ new_note)
475             elif isinstance(element, music21.chord.Chord):
476                 duration = element.duration.quarterLength
477                 expanded_duration = (1 / divisions) * duration
478                 offset = element.offset
479                 measure.remove(element)
480                 for i in range(divisions):
481                     new_chord =
482                         ↵ music21.chord.Chord(element.pitches)
483                     for note_in_chord in new_chord:
484                         note_in_chord.duration.quarterLength =
485                             ↵ expanded_duration
486                     measure.insert(offset + i * expanded_duration,
487                         ↵ new_chord)
488             elif isinstance(element, music21.note.Rest):
489                 duration = element.duration.quarterLength
490                 expanded_duration = (1 / divisions) * duration
491                 offset = element.offset

```

Web Application

This project features a Python-based web application that facilitates website creation and integrates a Python library for web functionalities. The following scripts are included:

- `app.py`: Primary script for launching the web application.
- `melody_api.py`: Implements the API endpoints for handling requests related to the melody generation system.
- `dynamic_system.py`: Contains the core logic for the dynamic system utilized within the application.

The corresponding code for each script is presented below:

app.py

```
1 import streamlit as st
2 import os
3 from melody_api import Melody
4 from dynamic_system.dynamic_system import *
5 import streamlit.components.v1 as components
6
7
8 def main():
9     def expanded_index_function(n):
10         if n == 2:
11             index = 0
12         elif n == 4:
13             index = 1
14         elif n == 8:
15             index = 2
16         elif n == 16:
17             index = 3
18         return index
19
20     if 'x' not in st.session_state:
21         st.session_state['x'] = 1.01
22     if 'y' not in st.session_state:
23         st.session_state['y'] = 1.01
24     if 'z' not in st.session_state:
25         st.session_state['z'] = 1.01
26     if 'sigma' not in st.session_state:
27         st.session_state['sigma'] = 10.0
28     if 'rho' not in st.session_state:
29         st.session_state['rho'] = 28.0
30     if 'beta' not in st.session_state:
31         st.session_state['beta'] = 2.67
32     if 'method' not in st.session_state:
33         st.session_state['method'] = "Classic"
34     if 'option' not in st.session_state:
35         st.session_state['option'] = 2
36     if 'sequence_select' not in st.session_state:
37         st.session_state['sequence_select'] = 0
38     if 'uploaded_file' not in st.session_state:
39         st.session_state['uploaded_file'] = None
40     if 'save_path' not in st.session_state:
41         st.session_state['save_path'] = None
42     if 'method_index' not in st.session_state:
43         st.session_state['method_index'] = 0
44     if 'expanded_index' not in st.session_state:
45         st.session_state['expanded_index'] = 0
46     if 'html_component' not in st.session_state:
47         st.session_state['html_component'] = None
48     if 'original_path' not in st.session_state:
49         st.session_state['original_path'] = None
```

```

50     if 'new_path' not in st.session_state:
51         st.session_state['new_path'] = None
52     if 'midi_new_download_path' not in st.session_state:
53         st.session_state['midi_new_download_path'] = None
54     if 'pdf_original_path' not in st.session_state:
55         st.session_state['pdf_original_path'] = None
56     if 'pdf_new_path' not in st.session_state:
57         st.session_state['pdf_new_path'] = None
58     if 'generate_clicked' not in st.session_state:
59         st.session_state['generate_clicked'] = False
60     if 'complete_download' not in st.session_state:
61         st.session_state['complete_download'] = False
62     if 'original_pdf_bytes' not in st.session_state:
63         st.session_state['original_pdf_bytes'] = None
64     if 'new_midi_bytes' not in st.session_state:
65         st.session_state['new_midi_bytes'] = None
66     if 'new_pdf_bytes' not in st.session_state:
67         st.session_state['new_pdf_bytes'] = None
68     if 'track_list' not in st.session_state:
69         st.session_state['track_list'] = None
70     if 'backup_track_list' not in st.session_state:
71         st.session_state['backup_track_list'] = None
72     if 'under_process' not in st.session_state:
73         st.session_state['under_process'] = None
74     if 'file_exist' not in st.session_state:
75         st.session_state['file_exist'] = 0
76
77     st.set_page_config(layout="wide")
78
79 # Initialize Melody object
80 my_melody = Melody()
81
82 # Page layout
83 left_col, right_col = st.columns([1, 3])
84
85 # MIDI file count variable
86 file_count = 0
87
88 # Left Column: Generate Music Button and File Upload
89 with left_col:
90
91     st.markdown(
92         '<h1 style="font-size:30px;">Variate Music</h1>',
93         unsafe_allow_html=True
94     )
95
96     uploaded_file = st.file_uploader("Upload a MIDI file",
97                                     type=["mid", "midi"])
98
99     CSS = '''
<style>

```

```

100     [data-testid='stFileUploader'] {
101         width: max-content;
102     }
103     [data-testid='stFileUploader'] section {
104         padding: 0;
105         float: left;
106     }
107     [data-testid='stFileUploader'] section > input + div {
108         display: none;
109     }
110     [data-testid='stFileUploader'] section + div {
111         display: none;
112     }
113 </style>
114 ''
115
116 st.markdown(css, unsafe_allow_html=True)
117
118 # Only update session state if a new file is uploaded
119
120 if uploaded_file is not None:
121     if st.session_state['file_exist'] == 1:
122         print(f"now_file = {uploaded_file.name}")
123         print(f"exist_file =
124             {st.session_state['uploaded_file'].name}")
125         if (uploaded_file.name !=
126             st.session_state['uploaded_file'].name):
127             st.session_state['track_list'] = None
128             st.session_state['backup_track_list'] = None
129             st.session_state['uploaded_file'] = uploaded_file
130             st.session_state['file_exist'] = 1
131
132 if 'uploaded_file' in st.session_state and
133     st.session_state['uploaded_file'] is not None:
134     st.write("Uploaded file:",
135             st.session_state['uploaded_file'].name)
136
137     save_directory = os.path.join("static", "midi_file")
138     os.makedirs(save_directory, exist_ok=True)
139
140     if ' ' in str(st.session_state['uploaded_file'].name):
141         st.warning('Warning: We can\'t save your MIDI file
142             because your MIDI file name has spacing. Please
143             fill the spacing with "_"')
144     else:
145         save_path = os.path.join(save_directory,
146             st.session_state['uploaded_file'].name)
147
148         # Reset file pointer to the beginning
149         st.session_state['uploaded_file'].seek(0)

```

```

144     # Save the uploaded file locally
145     with open(save_path, "wb") as f:
146         f.write(st.session_state['uploaded_file'].read())
147     st.success("File saved successfully!")
148
149     # Reset file pointer again to read for MIDI processing
150     st.session_state['uploaded_file'].seek(0)
151
152     try:
153         # Load MIDI data
154         # midi_data = mido.MidiFile(file=BytesIO(st.session_state['uploaded_file'].read()))
155         # st.title("MIDI File Information")
156         # st.write(f"Number of Tracks: {len(midi_data.tracks)}")
157
158         # Load into your Melody object
159         st.session_state['save_path'] = save_path
160         my_melody.load(path=st.session_state['save_path'])
161     except Exception as e:
162         st.error(f"Failed to read MIDI file: {e}")
163     generate_button = st.button('Variate')
164
165     # Right Column: Tabs for Results and Settings
166     with right_col:
167         tabs = st.tabs(["Results", "Settings"])
168
169         # Results Tab
170         with tabs[0]:
171             if generate_button:
172                 st.session_state['generate_clicked'] = True
173
174             if st.session_state['generate_clicked'] == False:
175                 st.image("intro.jpg")
176                 st.write("Start creating your variation by simply
177                         uploading a MIDI file and clicking \"Variate
178                         Music!\"")
179
180             if 'uploaded_file' in st.session_state and
181             st.session_state['uploaded_file'] is not None:
182                 if generate_button == True:
183                     st.snow()
184                     st.session_state['complete_download'] = False
185
186                     st.write("Waiting for our algorithm to generate a
187                             new variation!")
188                     my_melody.set_dynamic_sequence(sequence=st.session_state['sequence_select'])
189                     key_list = my_melody.fit(original_initial_conditions,
190                         n=[st.session_state['x'],
191                         st.session_state['y'], st.session_state['z']],
192                         )

```

```

186             method=st.session_state['method'],
187             ↵ divisions=st.session_state['option'],
188             dynamic=le, d=st.session_state['sigma'],
189             ↵ r=st.session_state['rho'],
190             ↵ b=st.session_state['beta'])
191     if st.session_state['track_list'] == None:
192         st.session_state['track_list'] = [key for key
193             ↵ in key_list]
194     st.session_state['backup_track_list'] = [key
195             ↵ for key in key_list]
196     my_melody.variate(new_initial_condition=[1.5, 1.5,
197             ↵ 1.5], track=st.session_state['track_list'])
198
199     original_path, new_path, midi_new_download_path =
200             ↵ my_melody.export("midi")
201
202     st.session_state['original_path'] = original_path
203     st.session_state['new_path'] = new_path
204
205     st.session_state['midi_new_download_path'] =
206             ↵ midi_new_download_path
207     with open(st.session_state['midi_new_download_path'],
208             ↵ ')', "rb") as file:
209         st.session_state['new_midi_bytes'] =
210             ↵ file.read()
211
212     pdf_original_path, pdf_new_path =
213             ↵ my_melody.export("pdf")
214
215     st.session_state['pdf_original_path'] =
216             ↵ pdf_original_path
217     with open(st.session_state['pdf_original_path'],
218             ↵ "rb") as file:
219         st.session_state['original_pdf_bytes'] =
220             ↵ file.read()
221
222     st.session_state['pdf_new_path'] = pdf_new_path
223     with open(st.session_state['pdf_new_path'], "rb")
224             ↵ as file:
225         st.session_state['new_pdf_bytes'] = file.read()
226
227     st.session_state['html_component'] = True
228
229
230 # MIDI Player and Visualizer
231 if 'html_component' in st.session_state and
232     ↵ st.session_state['html_component'] is not None:
233     components.html(
234         f"""
235         <h1 style="font-size:30px;">Original Music</h1>
236         <center>
237             <midi-player

```

```

221         src="{st.session_state['original_path']}""
222         sound-font visualizer="#myStaffVisualizer1">
223     </midi-player>
224
225     <midi-visualizer type="staff" id="myStaffVisualizer1"
226         src="{st.session_state['original_path']}">
227     </midi-visualizer>
228     </center>
229     <script
230         src="https://cdn.jsdelivr.net/combine/npm/tone@14.1.
231         7.58,npm/@magenta/music@1.23.1/es6/core.js,npm/foc
232         us-visible@5,npm/html-midi-player@1.5.0"></script>
233     """,
234     height=300)
235
236     original_col1, original_col2, original_col3,
237     ↵ original_col4, original_col5 =
238     ↵ st.columns([1,1,2,1,1])
239     with original_col3:
240         st.download_button(
241             label="Download Original Music Sheet",
242             data=st.session_state['original_pdf_bytes'],
243             file_name=str(st.session_state['uploaded_file']
244             ↵ ].name) + "_original.pdf",
245             mime="application/pdf"
246         )
247
248     components.html(
249         f"""
250         <h1 style="font-size:30px;">New Variation Music</h1>
251         <center>
252             <midi-player
253                 src="{st.session_state['new_path']}"
254                 sound-font visualizer="#myStaffVisualizer2">
255             </midi-player>
256
257             <midi-visualizer type="staff" id="myStaffVisualizer2"
258                 src="{st.session_state['new_path']}">
259             </midi-visualizer>
260             </center>
261             <script
262                 src="https://cdn.jsdelivr.net/combine/npm/tone@14.1.
263                 7.58,npm/@magenta/music@1.23.1/es6/core.js,npm/foc
264                 us-visible@5,npm/html-midi-player@1.5.0"></script>
265             """,
266             height=300
267         )
268
269     new_col1, new_col2 = st.columns(2)
270     with new_col1:
271         st.download_button(

```

```

263         label="Download MIDI File",
264         data=st.session_state['new_midi_bytes'],
265         file_name=str(st.session_state['uploaded_file']
266             .name) + "_new.mid",
267         mime="audio/midi"
268     )
269     with new_col2:
270         st.download_button(
271             label="Download New Music Sheet",
272             data=st.session_state['new_pdf_bytes'],
273             file_name=str(st.session_state['uploaded_file']
274                 .name) + "_new.pdf",
275             mime="application/pdf"
276         )
277     st.session_state['complete_download'] = True
278     st.write("Complete!")
279
280
281 # Settings Tab
282 with tabs[1]:
283     st.title("Settings")
284
285     equation = st.selectbox(
286         "Choose Chaotic Systems",
287         ("Lorenz system", "Chaotic system"),
288         index=0,
289         placeholder="Choose Equation"
290     )
291
292 # Actions for Lorenz system
293 if equation == "Lorenz system":
294     # Variable selection
295     variable_Lorenz = st.selectbox(
296         "Select variable to display",
297         options=("x", "y", "z"),
298         index=0
299     )
300
301 # Create a two-column layout
302 col1, col2 = st.columns(2)
303
304 with col1:
305     # Display Variables
306     st.markdown("### Variables")
307
308     # Sliders for variables
309     st.session_state['x'] = st.slider(
310         "x",
311         min_value=-5.0,
312         max_value=5.0,
313         value=1.01,

```

```

312             step=0.01
313         )
314     st.session_state['y'] = st.slider(
315         "y",
316         min_value=-5.0,
317         max_value=5.0,
318         value=1.01,
319         step=0.01
320     )
321     st.session_state['z'] = st.slider(
322         "z",
323         min_value=-5.0,
324         max_value=5.0,
325         value=1.01,
326         step=0.01
327     )
328
329     # Set the variable based on selection
330     if variable_Lorenz == "x":
331         st.session_state['sequence_select'] = 0
332     elif variable_Lorenz == "y":
333         st.session_state['sequence_select'] = 1
334     elif variable_Lorenz == "z":
335         st.session_state['sequence_select'] = 2
336
337     with col2:
338         # Display Parameters
339         st.markdown("### Parameters")
340
341         # Sliders for parameters
342         st.session_state['sigma'] = st.slider(
343             r"\sigma",
344             min_value=0.0,
345             max_value=100.0,
346             value=10.0,
347             step=0.01
348         )
349         st.session_state['rho'] = st.slider(
350             r"\rho",
351             min_value=0.0,
352             max_value=100.0,
353             value=28.0,
354             step=0.01
355         )
356         st.session_state['beta'] = st.slider(
357             r"\beta",
358             min_value=0.0,
359             max_value=100.0,
360             value=2.67,
361             step=0.01
362     )

```

```

363
364     # Check Chaotic Parameters
365     if st.button('Check Parameters'):
366         my_melody.load(path=st.session_state['save_path'])
367         my_melody.set_dynamic_sequence(sequence=st.session_
368             ↵ _state['sequence_select']))
369         my_melody.fit(initial_condition=[st.session_state[_
370             ↵ 'x'], st.session_state['y'], st.session_state['z']_
371             ↵ ']], dynamic=le, d=st.session_state['sigma'],
372             ↵ r=st.session_state['rho'],
373             ↵ b=st.session_state['beta']))
374         checker = my_melody.is_chaotic()
375         if str(checker) == 'True':
376             st.write(f"Chaotic parameter")
377         else:
378             st.warning('Parameter not chaotic')
379
380
381
382
383
384
385
386
387     # Additional Options
388     st.session_state['method'] = st.selectbox(
389         "Additional Options",
390         ("Classic", "Expanded"),
391         index=st.session_state['method_index'],
392         placeholder="Choose Equation",
393     )
394
395     if st.session_state['method'] == "Classic":
396         st.session_state['method_index'] = 0
397         st.session_state['option'] = None
398
399     elif st.session_state['method'] == "Expanded":
400         st.session_state['method_index'] = 1
401         st.session_state['option'] = st.selectbox(
402             "Expanded Number (Example: If you choose 4, our
403                 ↵ algorithm will expand 1 note into 4 notes in
404                 ↵ your MIDI file)",
405             (2, 4, 8, 16),
406             index=st.session_state['expanded_index']),

```

```

405         )
406     st.session_state['expanded_index'] =
407         ↳ expanded_index_function(st.session_state['option'])
408
409 if __name__ == "__main__":
410     main()

```

melody_api.py

```

1  import music21
2  from music21 import stream, midi, scale, converter, instrument
3  from datetime import datetime as dt
4  import numpy as np
5  import subprocess
6  import os
7  import pathlib
8
9  class Melody:
10
11     def __init__(self):
12         self.tracks = {}
13         self.maps = {}
14         self.dynamic = None
15
16         # Main Instance Attributes
17         self.original_midi_data = None
18         self.new_midi_data = None
19
20         # Additional Instance Attributes
21         self.newtracks = {}
22         self.main_stream = None
23         self.new_stream = None
24         self.duration = {}
25         self.new_duration = {}
26         self.initial_condition = None
27         self.dynamic_sequence = 0
28         self.notetrack = {}
29         self.tracks_index = {}
30         self.dynamic_parameter = {}
31
32         # Support Instance Attributes
33         self.method = None
34         self.divisions = None
35         self.backtrack_expand_index = []
36
37         # Path
38         self.lilypond_path =
39             ↳ str(pathlib.Path("__file__").parent.resolve()) +
40             ↳ "\\\lilypond-2.24.4\\bin\\lilypond.exe"
41
42         # etc.

```

```

41     self.filename = None
42     self.dummy1 = None
43     self.dummy2 = None
44     self.dummy3 = None
45     self.dummy4 = None
46
47 ######
48 # Main Method
49 #####
50
51 def load(self, path):
52     midi_file = midi.MidiFile()
53     midi_file.open(path)
54     midi_file.read()
55     midi_file.close
56
57     self.main_stream = midi.translate.midiFileToStream(midi_file)
58     self.new_stream = midi.translate.midiFileToStream(midi_file)
59
60     self.original_midi_data =
61         ↳ midi.translate.midiFileToStream(midi_file)
62     self.new_midi_data = midi.translate.midiFileToStream(midi_file)
63
64     self.dummy1 = midi.translate.midiFileToStream(midi_file)
65     self.dummy2 = midi.translate.midiFileToStream(midi_file)
66     self.dummy3 = midi.translate.midiFileToStream(midi_file)
67     self.dummy4 = midi.translate.midiFileToStream(midi_file)
68
69     self.filename = str(pathlib.Path(path).stem)
70     #return self.tracks
71
72 def fit(self, original_initial_condition, dynamic,
73        ↳ method="Classic", divisions=None, *args, **kwargs):
74     self.original_initial_condition = original_initial_condition
75     self.dynamic_parameter.update(*args, **kwargs)
76     self.dynamic = dynamic
77     self.method = method
78     self.divisions = divisions
79
80     if self.method == "Classic":
81         for index in range(len(self.main_stream)):
82             instruments = self.main_stream[index][0].getElementsByJ
83             ↳ Class(instrument.Instrument)
84             if instruments:
85                 instrument_name =
86                     ↳ str(instruments[0].instrumentName) + "_" +
87                     ↳ str(index)
88             else:
89                 instrument_name = "track_" + str(index)
90             music_list = []
91             duration_list = []

```

```

87     for part in self.main_stream[index]:
88         for n in part.flat.notesAndRests:
89             dur = n.duration.quarterLength
90             if n.isRest:
91                 note_num = "<REST>"
92                 music_list.append(note_num)
93                 duration_list.append(dur)
94             else:
95                 if len(n.pitches) == 1:
96                     note_num = n.pitches[0].midi
97                     music_list.append(note_num)
98                     duration_list.append(dur)
99                 else:
100                     note_num = sorted(set([p.midi for p in
101                         ↵ n.pitches]))
102                     music_list.append(list(note_num))
103                     duration_list.append(dur)
104             string_music_list = [element for element in music_list]
105             for i in range(len(music_list)):
106                 try:
107                     string_music_list[i] =
108                         ↵ self.note_converter(music_list[i])
109                 except TypeError:
110                     for j in range(len(music_list[i])):
111                         string_music_list[i][j] =
112                             ↵ self.note_converter(music_list[i][j])
113
114             time, solution = self.rk4(time=[0,
115                 ↵ 0.01*(len(music_list) - 1)], f=self.dynamic,
116                 ↵ ini=self.original_initial_condition,
117                 ↵ **self.dynamic_parameter)
118             main_sequence = solution[:, self.dynamic_sequence]
119             #self.original_time = time
120             # self.original_trajectory = main_sequence
121
122             self.notetrack.update({instrument_name:string_music_li
123                 ↵ st})
124             self.duration.update({instrument_name:duration_list})
125             self.tracks.update({instrument_name:main_sequence})
126             self.tracks_index.update({instrument_name:index})
127             for key in self.tracks.keys():
128                 dummy_list = []
129                 for i in range(len(self.notetrack[key])):
130                     dummy_list.append([self.tracks[key][i],
131                         ↵ self.notetrack[key][i]])
132                 self.maps.update({key:dummy_list})
133
134             elif self.method == "Expanded":
135                 for index in range(len(self.main_stream)):
136                     instruments = self.main_stream[index][0].getElementsByJ
137                         ↵ Class(instrument.Instrument)
138                     if instruments:

```

```

129         instrument_name =
130             ↵ str(instruments[0].instrumentName) + "_" +
131             ↵ str(index)
132     else:
133         instrument_name = "track_" + str(index)
134     duration_list = []
135     for part in self.main_stream[index]:
136         for n in part.flat.notesAndRests:
137             dur = n.duration.quarterLength
138             if n.isRest:
139                 duration_list.append(dur)
140             else:
141                 if len(n.pitches) == 1:
142                     duration_list.append(dur)
143                 else:
144                     duration_list.append(dur)
145     self.duration.update({instrument_name:duration_list})
146     for index in range(len(self.main_stream)):
147         self.expand_note_durations(s=self.main_stream[index],
148             ↵ divisions=self.divisions)
149         instruments = self.main_stream[index][0].getElementsBy_
150             ↵ Class(instrument.Instrument)
151         if instruments:
152             instrument_name =
153                 ↵ str(instruments[0].instrumentName) + "_" +
154                 ↵ str(index)
155         else:
156             instrument_name = "track_" + str(index)
157     music_list = []
158     #duration_list = []
159     for part in self.main_stream[index]:
160         for n in part.flat.notesAndRests:
161             dur = n.duration.quarterLength
162             if n.isRest:
163                 note_num = "<REST>"
164                 music_list.append(note_num)
165                 #duration_list.append(dur)
166             else:
167                 if len(n.pitches) == 1:
168                     note_num = n.pitches[0].midi
169                     music_list.append(note_num)
170                     #duration_list.append(dur)
171                 else:
172                     note_num = sorted(set([p.midi for p in
173                         ↵ n.pitches]))
174                     music_list.append(list(note_num))
175                     #duration_list.append(dur)
176     string_music_list = [element for element in music_list]
177     for i in range(len(music_list)):
178         try:

```

```

172             string_music_list[i] =
173                 ↵ self.note_converter(music_list[i]))
174         except TypeError:
175             for j in range(len(music_list[i])):
176                 string_music_list[i][j] =
177                     ↵ self.note_converter(music_list[i][j]))
178     self.backtrack_expand_index = [index for index in
179         ↵ range(0, len(music_list) - 1, self.divisions)]
180     time, solution = self.rk4(time=[0,
181         ↵ 0.01*(len(music_list) - 1)], f=self.dynamic,
182         ↵ ini=self.original_initial_condition,
183         ↵ **self.dynamic_parameter)
184     main_sequence = solution[:, self.dynamic_sequence]
185     #self.original_time = time
186     # self.original_trajectory = main_sequence
187
188     self.notetrack.update({instrument_name:string_music_li
189         ↵ st})
190     self.tracks.update({instrument_name:main_sequence})
191     self.tracks_index.update({instrument_name:index})
192     for key in self.tracks.keys():
193         dummy_list = []
194         for i in range(len(self.notetrack[key])):
195             dummy_list.append([self.tracks[key][i],
196                 ↵ self.notetrack[key][i]])
197         self.maps.update({key:dummy_list})
198     else:
199         raise Exception("Out of Option!")
200
201     #return self.maps
202     return self.tracks.keys()
203
204
205
206     def variate(self, new_initial_condition, track=None,
207         ↵ criteria="right", add_note=0):
208         self.new_initial_condition = new_initial_condition
209         if track == None:
210             track = list(self.tracks.keys())
211         if add_note != 0:
212             for key in track:
213                 for each_note in range(add_note):
214                     # Access parts from the stream
215                     parts = self.new_stream.getElementsByClass('Part')
216                     target_part = parts[self.tracks_index[key]]
217
218                     # Determine the last measure in the part
219                     if target_part.measure(-1):
220                         last_measure = target_part.measure(-1) # Get
221                             ↵ the last measure
222                     else:
223                         # If no measures exist, create the first one
224                         last_measure = music21.stream.Measure()

```

```

213         target_part.append(last_measure)
214
215     # Create the new note
216     new_note = music21.note.Note('C4')
217     duration_len = len(self.duration[key])
218     new_note.quarterLength =
219         self.duration[key][each_note % duration_len]
220
221     # Append the new note to the last measure
222     last_measure.append(new_note)
223     #print(target_part[-1].show("text"))
224     #print("-"*100)
225
226 if self.method == "Expanded":
227     for index in range(len(self.new_stream)):
228         self.expand_note_durations(s=self.new_stream[index],
229             divisions=self.divisions)
230         instruments = self.new_stream[index][0].getElementsByClass(
231             instrument.Instrument)
232         if instruments:
233             instrument_name =
234                 str(instruments[0].instrumentName) + "_" +
235                 str(index)
236         else:
237             instrument_name = "track_" + str(index)
238         duration_list = []
239         for part in self.new_stream[index]:
240             for n in part.flat.notesAndRests:
241                 dur = n.duration.quarterLength
242                 duration_list.append(dur)
243             self.new_duration.update({instrument_name:duration_list,
244                 t})
245         #self.duration = self.new_duration
246
247 for key in track:
248     sorted_tracks = sorted(self.maps[key], key=lambda x: x[0])
249     if self.method == "Classic":
250         dummy_list = [None]*(len(self.tracks[key]) + add_note)
251         time, solution = self.rk4(time=[0,
252             0.01*(len(dummy_list) - 1)], f=self.dynamic,
253             ini=self.new_initial_condition,
254             **self.dynamic_parameter)
255         main_sequence = solution[:, self.dynamic_sequence]
256     elif self.method == "Expanded":
257         dummy_list = [None]*(len(self.tracks[key]) +
258             add_note)*self.divisions
259         time, solution = self.rk4(time=[0,
260             0.01*(len(dummy_list) - 1)], f=self.dynamic,
261             ini=self.new_initial_condition,
262             **self.dynamic_parameter)
263         main_sequence = solution[:, self.dynamic_sequence]

```

```

251         # self.new_time = time
252         # self.new_trajectory = main_sequence
253     else:
254         raise Exception("Out of Option!")
255
256     if criteria == "right":
257         for i in range(len(main_sequence)) if len(main_sequence)
258             == len(dummy_list) else len(main_sequence) - 1):
259             for j in range(len(sorted_tracks)):
260                 if main_sequence[i] <= sorted_tracks[j][0]:
261                     dummy_list[i] = sorted_tracks[j][1]
262                     break
263                 elif main_sequence[i] > sorted_tracks[-1][0]:
264                     dummy_list[i] = sorted_tracks[-1][1]
265                     break
266             elif criteria == "left":
267                 for i in range(len(main_sequence)) if len(main_sequence)
268                     == len(dummy_list) else len(main_sequence) - 1):
269                     for j in range(len(sorted_tracks)):
270                         if main_sequence[i] <= sorted_tracks[j][0]:
271                             dummy_list[i] = sorted_tracks[j - 1][1]
272                             break
273                         elif main_sequence[i] > sorted_tracks[-1][0]:
274                             dummy_list[i] = sorted_tracks[-1][1]
275                             break
276                         elif (main_sequence[i] <= sorted_tracks[0][0])
277                             and (j - 1 < 0):
278                             main_sequence[i] = sorted_tracks[0][0]
279                             break
280             else:
281                 raise Exception("Out of Option!")
282
283         self.newtracks.update({key:dummy_list})
284
285     def export(self, format_type):
286         exist_track = [track for track in self.newtracks.keys()]
287         exist_new_stream = self.new_midi_data
288         if self.method == "Classic":
289             for key in self.tracks.keys():
290                 if key in exist_track:
291                     index = 0
292                     for element in exist_new_stream[self.tracks_index[,
293                         key]].recurse():
294                         if isinstance(element, (music21.note.Note,
295                             music21.chord.Chord, music21.note.Rest)):
296                             replacement = self.newtracks[key][index]
297                             duration_value = self.duration[key][index]
298                             if replacement == '<REST>':
299                                 new_element = music21.note.Rest()
300                             elif isinstance(replacement, list):

```

```

296             new_element =
297                 ↵ music21.chord.Chord(replacement)
298         else:
299             new_element =
300                 ↵ music21.note.Note(replacement)
301         new_element.duration = music21.duration.Duration()
302             ↵ ration(duration_value)
303         element.activeSite.replace(element,
304             ↵ new_element)
305         print(f"{element} --> {replacement}")
306     if index < len(self.newtracks[key]) - 1:
307         index += 1
308     else:
309         break
310
311 elif self.method == "Expanded":
312     for key in self.tracks.keys():
313         if key in exist_track:
314             index = 0
315             for element in exist_new_stream[self.tracks_index[...],
316                 ↵ key]].recurse():
317                 if isinstance(element, (music21.note.Note,
318                     ↵ music21.chord.Chord, music21.note.Rest)):
319                     replacement = self.newtracks[key][self.backtrack_index[...],
320                         ↵ ktrack_expand_index[index]]
321                     duration_value = self.duration[key][index]
322                     if replacement == '<REST>':
323                         new_element = music21.note.Rest()
324                     elif isinstance(replacement, list):
325                         new_element =
326                             ↵ music21.chord.Chord(replacement)
327                     else:
328                         new_element =
329                             ↵ music21.note.Note(replacement)
330                     new_element.duration = music21.duration.Duration()
331                         ↵ ration(duration_value)
332                     element.activeSite.replace(element,
333                         ↵ new_element)
334                     #print(f"{element} --> {replacement}")
335                     if index < len(self.backtrack_expand_index):
336                         ↵ - 1:
337                         index += 1
338                     else:
339                         break
340
341 else:
342     raise Exception("Out of Option!")
343 current_path = pathlib.Path("__file__").parent.resolve()
344 date = str(dt.now().isoformat())[:-7]
345 filepath = str(current_path) + "\\" + "static" + "\\" +
346     ↵ "file_store" + "\\" + self.filename + date.replace(":", "-")
347 isExist = os.path.exists(filepath)

```

```

333     if not isExist:
334         os.makedirs(filepath)
335     original = self.original_midi_data
336     new = self.new_midi_data
337     streamlit_accept_path = "app" + "/" + "static" + "/" +
338     ↳ "file_store" + "/" + self.filename + date.replace(":", "-")
339     if format_type == "midi":
340         # original.metadata = music21.metadata.Metadata()
341         original.write('midi', filepath + "\\\" + self.filename +
342         ↳ '_original.mid')
343         original_path = streamlit_accept_path + "/" + self.filename
344         ↳ + '_original.mid'
345         # new.metadata = music21.metadata.Metadata()
346         if self.method == "Classic":
347             new.write('midi', filepath + "\\\" + self.filename +
348             ↳ '_new.mid')
349             new_path = streamlit_accept_path + "/" + self.filename
350             ↳ + '_new.mid'
351             midi_new_download_path = filepath + "\\\" +
352             ↳ self.filename + '_new.mid'
353             elif self.method == "Expanded":
354                 new.write('midi', filepath + "\\\" + self.filename +
355                 ↳ '_new_expanded.mid')
356                 new_path = streamlit_accept_path + "/" + self.filename
357                 ↳ + '_new_expanded.mid'
358                 midi_new_download_path = filepath + "\\\" +
359                 ↳ self.filename + '_new_expanded.mid'
360             return original_path , new_path, midi_new_download_path
361             elif format_type == "pdf":
362                 us = music21.environment.UserSettings()
363                 us['lilypondPath'] = self.lilypond_path
364
365                 original_path = filepath + "\\\" + self.filename +
366                 ↳ '_original.pdf'
367                 original.metadata = music21.metadata.Metadata()
368                 original.metadata.title = self.filename.replace("_", " ") +
369                 ↳ " Original"
370                 conv = music21.converter.subConverters.ConverterLilypond()
371                 original_filepath = filepath + "\\\" + self.filename +
372                 ↳ '_original'
373                 conv.write(original, fmt='lilypond', fp=original_filepath,
374                 ↳ subformats = ['pdf'])
375
376                 self.main_stream = self.dummy1
377                 self.original_midi_data = self.dummy2
378
379                 new_path = filepath + "\\\" + self.filename + '_new.pdf'
380                 new.metadata = music21.metadata.Metadata()
381                 new.metadata.title = self.filename.replace("_", " ") +
382                 ↳ " New"

```

```

369     new_conv =
370     ↵ music21.converter.subConverters.ConverterLilypond()
371     new_filepath = filepath + "\\\" + self.filename + '_new'
372     conv.write(new, fmt='lilypond', fp=new_filepath, subformats
373     ↵ = ['pdf'])
374
375     self.new_stream = self.dummy3
376     self.new_midi_data = self.dummy4
377     return original_path , new_path
378
379 ######
380 # Additional Method
381 ######
382
383 def set_dynamic_sequence(self, sequence):
384     self.dynamic_sequence = sequence
385
386 ######
387 # Support Method
388 ######
389
390 def rk4(self, time, ini, f, h=0.01, *args, **kwargs):
391     t = np.arange(time[0], time[1]+h, h)
392     row = len(t)
393     try:
394         var = len(ini)
395     except TypeError:
396         var = 1
397     x = np.zeros((row, var))
398     x[0, :] = ini
399     for j in range(0, len(t)-1):
400         k1 = f(t[j], x[j], *args, **kwargs)
401         k2 = f(t[j] + 0.5*h, x[j] + 0.5*h*k1, *args, **kwargs)
402         k3 = f(t[j] + 0.5*h, x[j] + 0.5*h*k2, *args, **kwargs)
403         k4 = f(t[j] + h, x[j] + h*k3, *args, **kwargs)
404         x[j+1, :] = x[j] + (h/6)*(k1 + 2*k2 + 2*k3 + k4)
405     return t, x
406
407 def is_chaotic(self):
408     T = [0, 100]
409
410     # Initial state and parameters
411     initial_state = np.array([1.0, 1.0, 1.0])
412
413     # Integrate system
414     #sol = solve_ivp(lorenz, t_span, initial_state, t_eval=t_eval,
415     ↵ rtol=1e-9, atol=1e-9)
416     time, sol = self.rk4(time=T, ini=initial_state, f=self.dynamic,
417     ↵ **self.dynamic_parameter)
418
419     # Perturb the initial state slightly

```

```

416     perturbed_state = np.array([1.00001, 1.0, 1.0])
417
418     #perturbed_sol = solve_ivp(lorenz, t_span, perturbed_state,
419     #                           t_eval=t_eval, rtol=1e-9, atol=1e-9)
420     time, perturbed_sol = self.rk4(time=T, ini=perturbed_state,
421     #                           f=self.dynamic, **self.dynamic_parameter)
422
423
424     # Compute divergence between trajectories
425     divergence = np.linalg.norm(sol - perturbed_sol, axis=1)
426
427
428     # Estimate Lyapunov exponent
429     divergence_rate = np.log(divergence + 1e-10) # Avoid log(0)
430     lyapunov_exponent = np.mean(divergence_rate[time > 10]) #
431     # Ignore transient phase
432
433     # Chaos detected if Lyapunov exponent is positive
434     return lyapunov_exponent > 0
435
436
437 def note_converter(self, midi_note):
438     """
439     This function convert single midi note value to readable
440     musical note.
441     Important parameter:
442     midi_note: midi note value;
443             60 ----> C4
444     """
445
446     if midi_note == '<REST>':
447         return '<REST>'
448     else:
449         if 0 <= midi_note <= 127:
450             pitches = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G',
451                         'G#', 'A', 'A#', 'B']
452
453             octave = (midi_note // 12) - 1
454             pitch_index = midi_note % 12
455
456             pitch = pitches[pitch_index]
457             return f'{pitch}{octave}'
458         else:
459             return "Invalid MIDI note value. Must be between 0 and
460             127."
461
462 def expand_note_durations(self, s, divisions):
463     for measure in s.getElementsByClass(stream.Measure):
464         for element in measure:
465             if isinstance(element, music21.note.Note):
466                 duration = element.duration.quarterLength
467                 expanded_duration = (1 / divisions) * duration
468                 offset = element.offset
469                 measure.remove(element)
470                 for i in range(divisions):
471

```

```

461             new_note = music21.note.Note(element.pitch)
462             new_note.duration.quarterLength =
463                 ↵ expanded_duration
464             measure.insert(offset + i * expanded_duration,
465                             ↵ new_note)
466             elif isinstance(element, music21.chord.Chord):
467                 duration = element.duration.quarterLength
468                 expanded_duration = (1 / divisions) * duration
469                 offset = element.offset
470                 measure.remove(element)
471                 for i in range(divisions):
472                     new_chord =
473                         ↵ music21.chord.Chord(element.pitches)
474                         for note_in_chord in new_chord:
475                             note_in_chord.duration.quarterLength =
476                                 ↵ expanded_duration
477                             measure.insert(offset + i * expanded_duration,
478                                         ↵ new_chord)
479             elif isinstance(element, music21.note.Rest):
480                 duration = element.duration.quarterLength
481                 expanded_duration = (1 / divisions) * duration
482                 offset = element.offset
483                 measure.remove(element)
484                 for i in range(divisions):
485                     new_rest = music21.note.Rest()
486                     new_rest.duration.quarterLength =
487                         ↵ expanded_duration
488                     measure.insert(offset + i * expanded_duration,
489                                     ↵ new_rest)

```

dynamic_system.py

```

1 import numpy as np
2
3 def le(t, xyz, *args, **kwargs):
4     """
5         *****
6         *****
7         "I didn't have any knowledge about Chaotic System" you say,
8         Nah Just use this one and change to any chaotic system you want
9         ↵ later.
10        Chaotic Trajectory parameter:
11        d: 10
12        r: 28
13        b: 8/3
14        *****
15        !!! This function relate with rk4 !!!
16        """
17
18        d, r, b = kwargs['d'], kwargs['r'], kwargs['b']
19        x = xyz[0]

```

```
17     y = xyz[1]
18     z = xyz[2]
19     xdot = d*(y - x)
20     ydot = x*(r - z) - y
21     zdot = x*y - b*z
22     return np.array([xdot, ydot, zdot])
```


Music Sheets

Pachelbel's Canon

Original Music Sheet

Johann Pachelbel Canon in D

The musical score consists of two staves of music. The top staff is in treble clef and the bottom staff is in bass clef. Both staves are in common time and key signature of D major (two sharps). The music is divided into measures numbered 1 through 35. The notation includes various note values such as eighth and sixteenth notes, and rests. Measure 1 starts with a rest followed by a sixteenth-note pattern. Measures 2-4 show a continuation of this pattern. Measures 5-8 introduce a new melodic line. Measures 9-12 continue the established patterns. Measures 13-16 show further development. Measures 17-20 introduce a new section with eighth-note patterns. Measures 21-24 continue this section. Measures 25-28 show a return to previous patterns. Measures 29-32 introduce a new section with sixteenth-note patterns. Measures 33-35 conclude the piece.

2

39

43

47

51

55

60

65

70

74

78

83

88

93

98

Music engraving by LilyPond 2.24.4—www.lilypond.org

New Variation from Mapping β

Our Variation

2

35

39

42

45

48

51

55

60

65

70

74

78

83

88

New Variation from Mapping β

Our Variation

The musical score for "Our Variation" is a two-staff composition in G major (two sharps) and 2/4 time. The top staff (Treble) begins with a whole note followed by a rest, then continues with a steady eighth-note pattern. The bottom staff (Bass) maintains a constant eighth-note pulse throughout. The piece is organized into six systems, each containing four measures. Measure numbers 1 through 30 are indicated on the left side of the score.

2

35

39

42

45

48

51

55

60

65

70

74

78

83

88

River Flows In You

Original Music Sheet

River Flows In You

1

5

8

11

14

17

20

2

24

27

30

34

40

3

51

52

53

54

55

56

57

58

59

60

Music engraving by LilyPond 2.24.4—www.lilypond.org

New Variation from Mapping β

Our Variation

The musical score consists of six staves of music. The first staff begins with a treble clef, a key signature of one sharp (G major), and a common time signature. It features a series of eighth-note patterns and grace notes. The second staff begins with a bass clef, a key signature of one sharp (G major), and a common time signature. It includes measures with quarter notes and eighth-note patterns. Subsequent staves continue this pattern, with measure numbers 4, 13, and 14 indicated. The music is characterized by its rhythmic complexity and harmonic richness, typical of a virtuosic piano variation.

2

18

23

26

31

3

41

44

47

50

New Variation from Mapping β

Our Variation

4

13

18

2

23

26

31

38

39

40

41

42

43

44

45

46

47

48

49

50