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
Deparment of Mathematics and Computer Science
Faculty of Science and Technology

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
Deparment of Mathematics and Computer Science
Faculty of Science and Technology

VER. EN20250313.0025

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

List of Figures

List of Tables

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 $\mathbb{N}_n := \{0, 1, 2, \ldots, n\}$, representing the first n natural numbers. A sequence of real numbers is written as $\{a_k\}_{k=0}^n = \{a_0, a_1, \ldots, a_n\}$, where each $a_k \in \mathbb{R}$ for $k = 0, 1, \ldots, n$. 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

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

The norm of a vector $v = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$ 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 $f : \mathbb{R} \to \mathbb{R}$, with its domain represented by $\mathrm{dom}(f)$. 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:

$$\begin{aligned} x_1'(t) &= f_1(t, x_1(t), x_2(t), \ldots, x_n(t)), \\ x_2'(t) &= f_2(t, x_1(t), x_2(t), \ldots, x_n(t)), \\ &\vdots \\ x_n'(t) &= f_n(t, x_1(t), x_2(t), \ldots, x_n(t)), \end{aligned} \tag{2.1}$$

where t is the independent variable defined on an interval I, $(x_1(t), x_2(t), \ldots, x_n(t)) \in \mathbb{R}^n$ are the dependent variables, and f_i are function from $I \times \mathbb{R}^n$ to $\mathbb{R}$ for $i = 1, 2, \ldots, n$. The notation $x_1'(t), x_2'(t), \ldots, x_n'(t)$ represents the first-order derivatives of $x_1(t), x_2(t), \ldots, x_n(t)$ 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 nth-order ordinary differential equation:

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

By introducing new variables:

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

we obtain the equivalent first-order system:

$$\begin{aligned} x_1'(t) &= x_2(t), \\ x_2'(t) &= x_3(t), \\ &\vdots \\ x_{n-1}'(t) &= x_n(t), \\ x_n'(t) &= f(t, x_1(t), x_2(t), \ldots, x_n(t)). \end{aligned}$$

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:

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

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 $\varphi : I \to \mathbb{R}^n$ 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 $\dot{\varphi}_1(t)$ 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 $\dot{\varphi}_2(t)$ 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:

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

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 $f : I \times \mathbb{R}^n \to \mathbb{R}^n$ is a given function. Additionally, $x(t_0)$ is an initial condition at time t_0.

Definition 2.3 ([7])**.** A function $\varphi : I \to \mathbb{R}^n$ 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 $\varphi(t)$:

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

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

$$f : D \subseteq R^n \to R^m$$

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 $f : \mathbb{R} \to \mathbb{R}$ 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 $f : I \times \mathbb{R}^n \to \mathbb{R}^n$ 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 $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$.

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:*

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

satisfies:

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

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 $x = 0$:

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

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,

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

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.

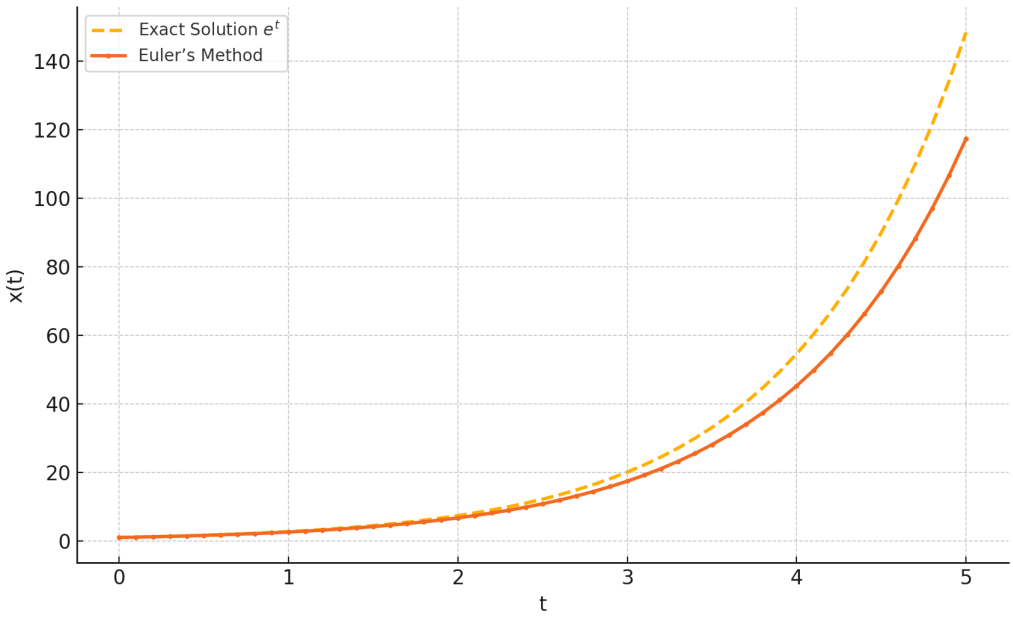


Figure 2.1: Example of Euler's Method.

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.
$t_0 = 0$, $x_0 = 1$.
$t_1 = 0.1$

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

$t_2 = 0.2$

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

$t_3 = 0.3$

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

$t_4 = 0.4$

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

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.

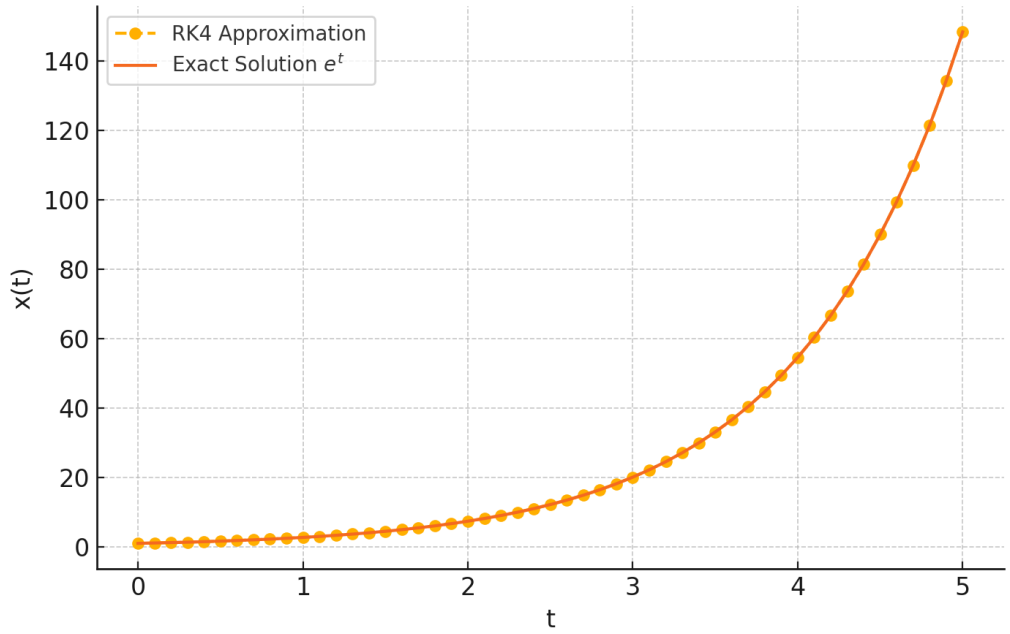


Figure 2.2: Example of the fourth-order Runge-Kutta method.

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 $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$.

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 $f : \mathbb{R}^n \to \mathbb{R}^m$ have components

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

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

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

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,

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

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*

$$\det(A - \lambda I) = 0 \tag{2.9}$$

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* $\dfrac{\partial f}{\partial x}(x)$ *be the Jacobian matrix for* (2.7) *evaluated at an equilibrium point* $\mathbf{x}^*$ *and* $\lambda_1, \lambda_2, \ldots, \lambda_n$ *be its eigenvalues.*

(i) If $Re(\lambda_i) < 0$ *for all* $i \in \{1, 2, \ldots, n\}$*, then the equilibrium point* $\mathbf{x}^*$ *is stable.*

(ii) If $Re(\lambda_i) > 0$ *for at least one* $i \in \{1, 2, \ldots, n\}$*, then the equilibrium point* $\mathbf{x}^*$ *is unstable.*

(iii) If $Re(\lambda_i) = 0$ *for at least one* $i \in \{1, 2, \ldots, n\}$*, 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 $\mathrm{Re}(\lambda_i) < 0$ for all eigenvalues, then the equilibrium point is stable.
- If $\mathrm{Re}(\lambda_i) > 0$ for at least one eigenvalue, then the equilibrium point is unstable.
- If $\mathrm{Re}(\lambda_i) = 0$ for at least one eigenvalue, the stability cannot be determined by linearization.

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

We can confirm the correctness of the solution using numerical methods. Here, we use the 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.

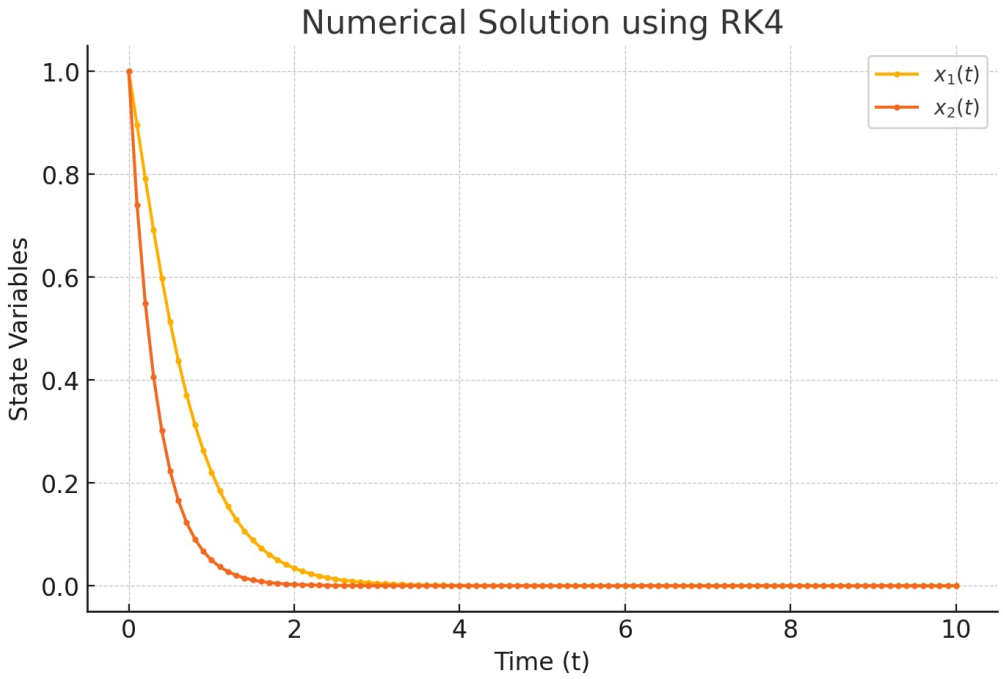


Figure 2.3: Numerical solution of the system (2.9) obtained using the RK4 method.

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 $\dot{x}_3 = 0$:

$$\sigma(x_2 - x_1) = 0, \tag{2.11}$$
$$\rho x_1 - x_2 - x_1 x_3 = 0, \tag{2.12}$$
$$x_1 x_2 - \beta x_3 = 0. \tag{2.13}$$

From (2.11), we obtain:

$$\begin{aligned} \sigma(x_2 - x_1) &= 0, \\ x_2 - x_1 &= 0, \\ x_2 &= x_1. \end{aligned}$$

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

$$\begin{aligned} \rho x_1 - x_1 - x_1 x_3 &= 0, \\ x_1(\rho - 1 - x_3) &= 0. \end{aligned}$$

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:

$$\begin{aligned} (0)x_2 - \beta x_3 &= 0, \\ -\beta x_3 &= 0, \\ x_3 &= 0. \end{aligned}$$

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:

$$\begin{aligned} x_1 x_2 - \beta(\rho - 1) &= 0, \\ x_1 x_1 &= \beta(\rho - 1), \\ x_1^2 &= \beta(\rho - 1), \\ x_1 &= \pm\sqrt{\beta(\rho - 1)}. \end{aligned}$$

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.

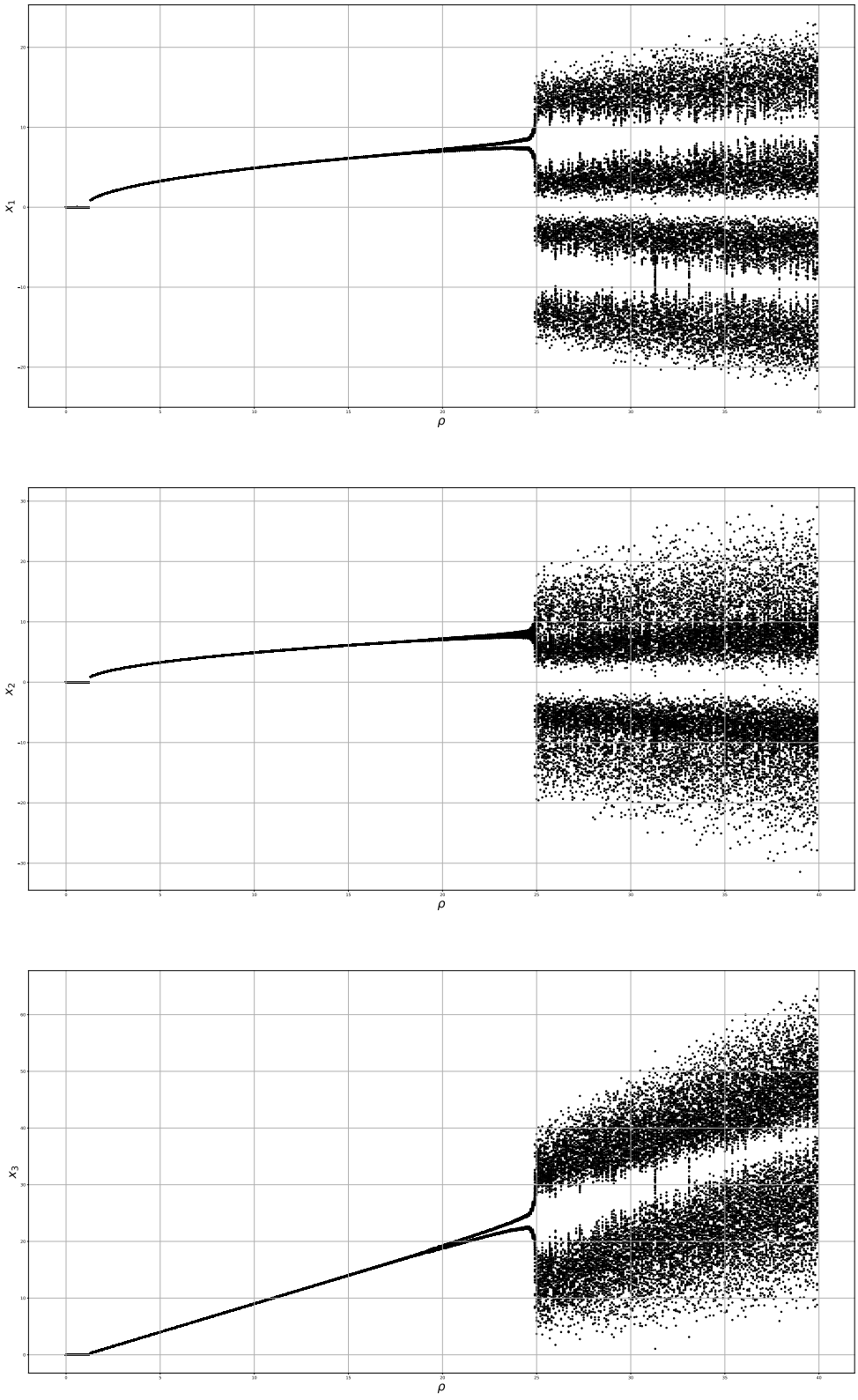


Figure 2.4: Bifurcation diagram of the Lorenz system for ρ ranging from 0 to 40.

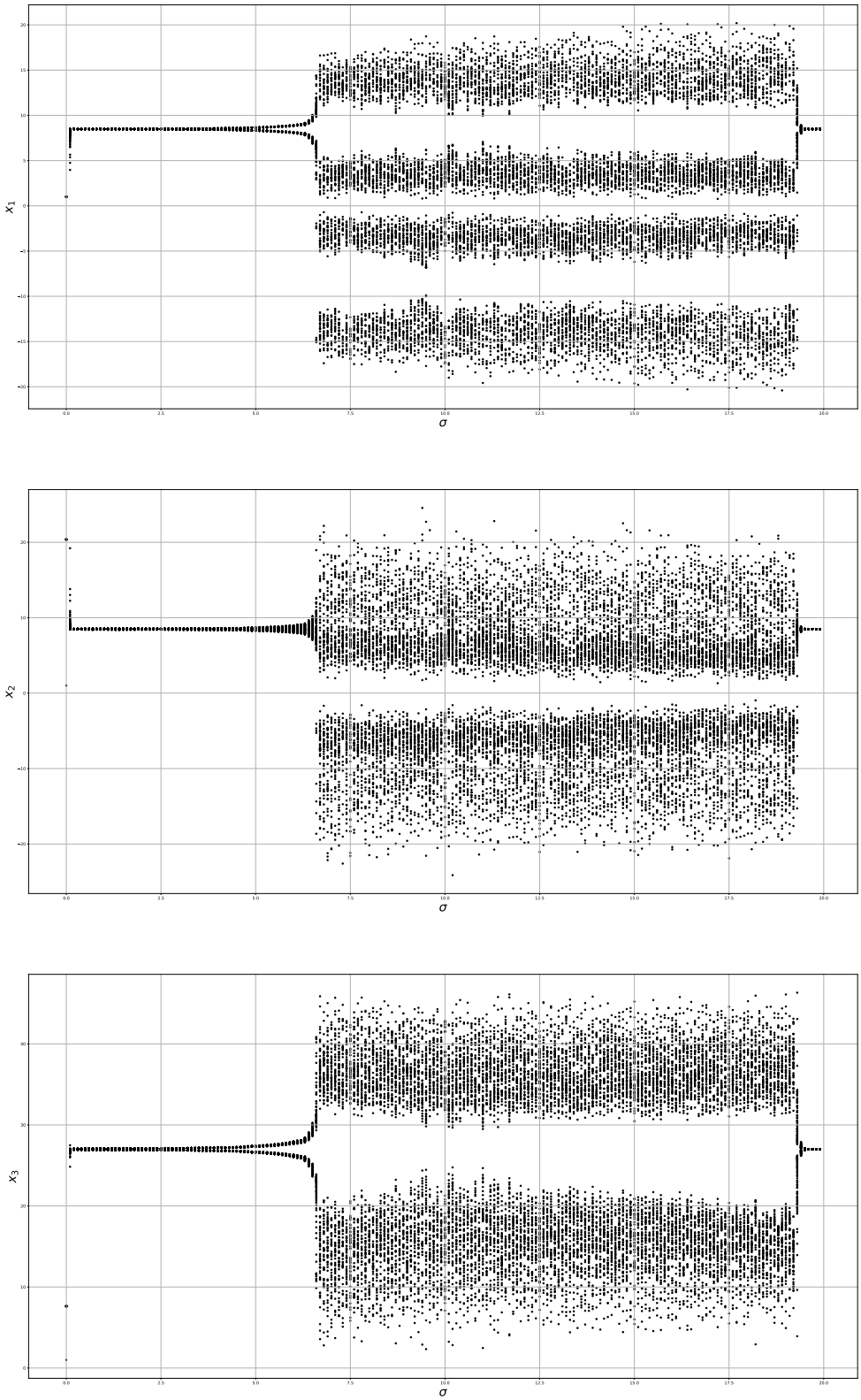


Figure 2.5: Bifurcation diagram of the Lorenz system for σ ranging from 0 to 20.

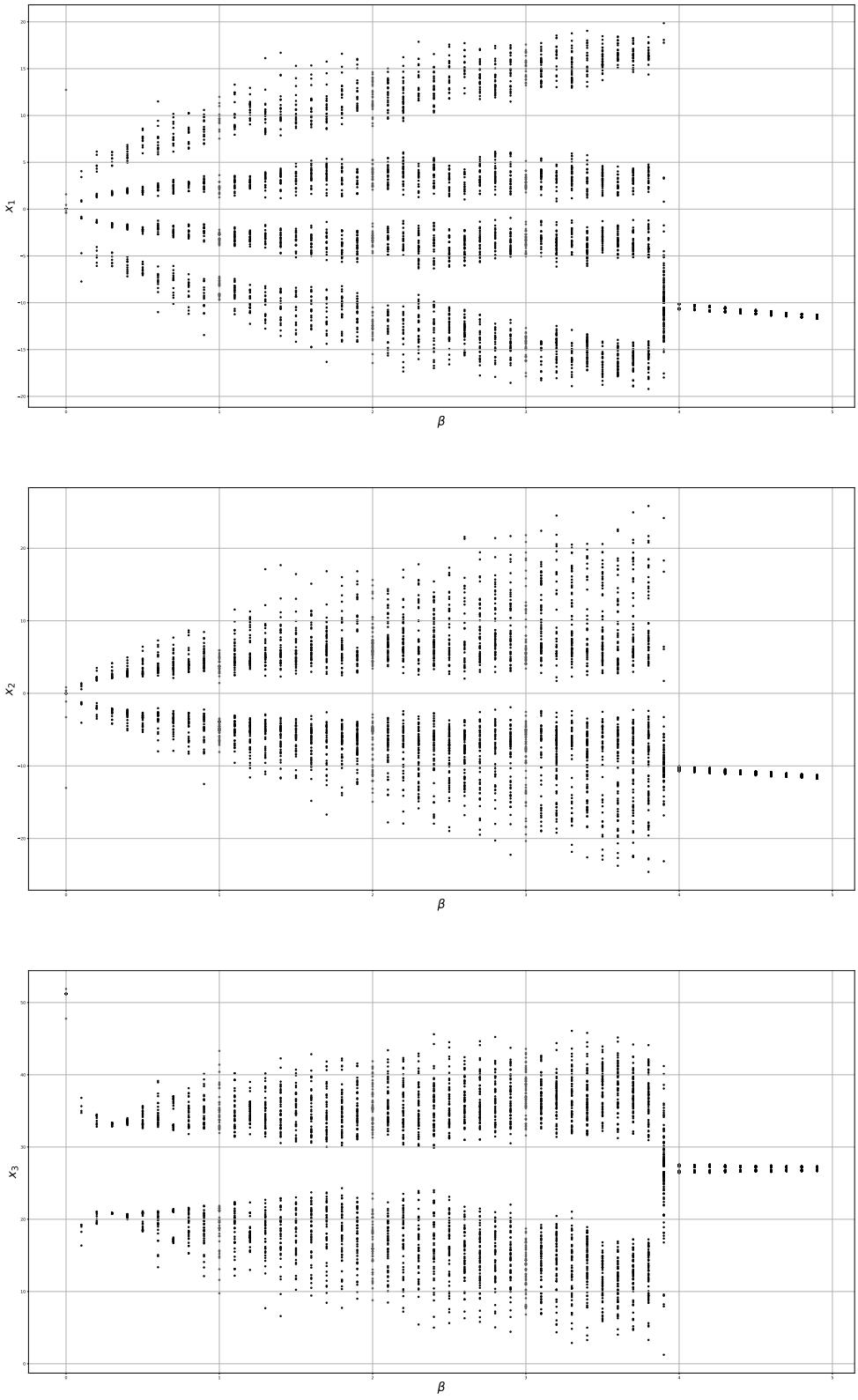
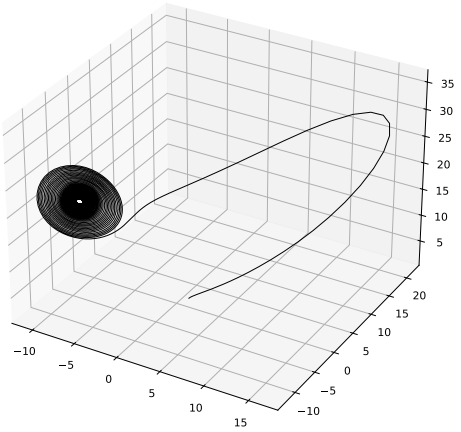
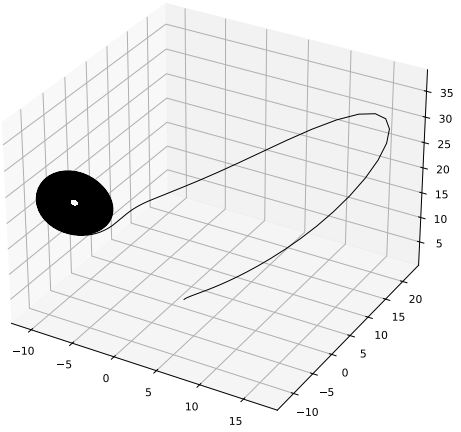


Figure 2.6: Bifurcation diagram of the Lorenz system for β ranging from 0 to 5.

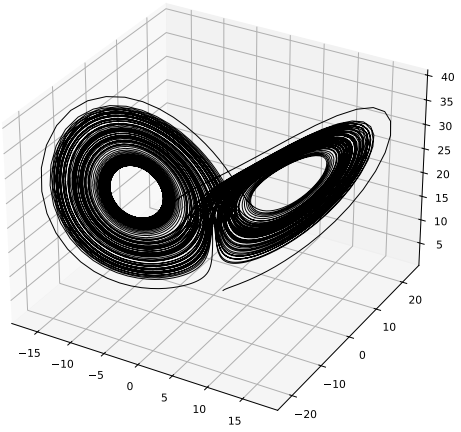
Let us consider an example to clarify why chaotic behavior occurs within certain ranges of the parameter values using Figure 2.4, where the values of x_1, x_2 and x_3 exhibit multiple points when $\rho \geq 24.95$. We simulate the Lorenz system with the initial condition $(1, 1, 1)$ and parameters $\rho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



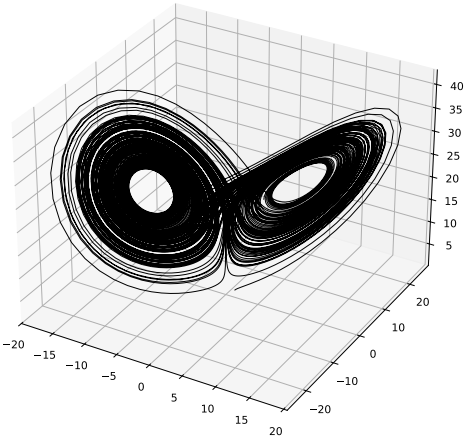
(a) $\rho = 22$



(b) $\rho = 23$



(c) $\rho = 24$



(d) $\rho = 25$

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:

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

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:

$$\begin{aligned} \lambda_1 &= 11.8277, \\ \lambda_2 &= -22.8277, \\ \lambda_3 &= -2.6667. \end{aligned}$$

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:

$$\begin{aligned} \lambda_1 &= -13.8545, \\ \lambda_2 &= 0.0939 + 10.1945i, \\ \lambda_3 &= 0.0939 - 10.1945i. \end{aligned}$$

   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:

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

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:

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

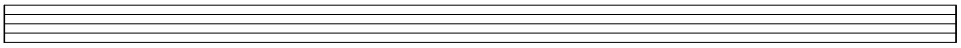
Example 2.27. The octaves are compared as follows:

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

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:

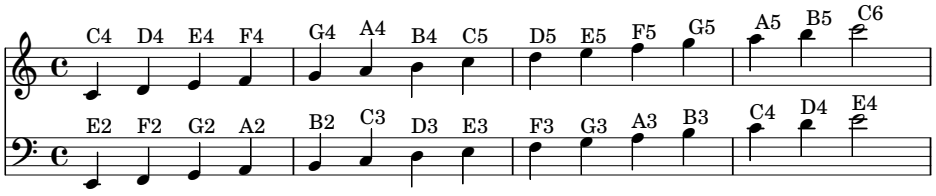


The clef symbol placed at the beginning of the staff indicates the pitch range of the notes. The two primary clefs are:

- Treble Clef (𝄞): Used for higher-pitched notes (e.g., violin, flute).
- Bass Clef (𝄢): 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 (𝄞), whereas the lower staff utilizes the Bass Clef (𝄢). In this example, let 𝅗𝅥 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

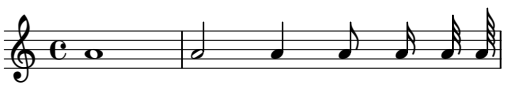
Duration refers to the length of time a pitch is played. In music theory, duration is typically measured in beats. Let us define one beat as one second for convenience. The standard note durations are as follows:

- The **whole note** (𝅝) is a musical symbol that represents a duration of 4 beats, equivalent to 4 seconds.
- The **half note** (𝅗𝅥) is a notation that indicates a playing time of 2 beats, which corresponds to 2 seconds.
- The **quarter note** (♩) denotes a duration of 1 beat, lasting for 1 second.
- The **eighth note** (♪) signifies a duration of $\frac{1}{2}$ of a beat, or $\frac{1}{2}$ of a second.
- The **sixteenth note** (𝅘𝅥𝅯) represents a duration of $\frac{1}{4}$ of a beat, corresponding to $\frac{1}{4}$ of a second.
- The **thirty-second note** (𝅘𝅥𝅰) 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 (𝄞):



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

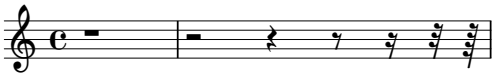
Rests

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 (𝄞):



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 (𝄽) lasting for 1 second, the eighth rest (𝄾) for $\frac{1}{2}$ second, the sixteenth rest (𝄿) for $\frac{1}{4}$ second, the thirty-second note (𝅀) 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 𝄻 has the longest duration.

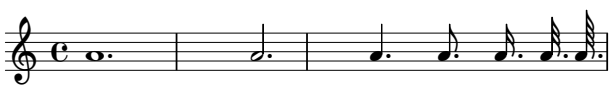
Dotted Notes

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** (𝅝.) represents a duration of 6 beats, equivalent to 6 seconds.
- The **dotted half note** (𝅗𝅥.) extends the half note's duration to 3 beats, corresponding to 3 seconds.
- The **dotted quarter note** (♩.) indicates a duration of 1.5 beats, lasting for 1.5 seconds.
- The **dotted eighth note** (♪.) signifies a duration of $\frac{3}{4}$ of a beat, or $\frac{3}{4}$ of a second.
- The **dotted sixteenth note** (𝅘𝅥𝅯.) represents a duration of $\frac{3}{8}$ of a beat, corresponding to $\frac{3}{8}$ of a second.
- The **dotted thirty-second note** (𝅘𝅥𝅰.) 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** (𝅘𝅥𝅱.) 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 (𝄞):



In standard notation, musical notes are read from left to right. The sequence begins with the dotted whole note (𝅝.), which has a duration of 6 seconds. Upon completion of the dotted whole note, the dotted half note (𝅗𝅥.) follows, indicating a duration of 3 seconds. This pattern continues with the dotted quarter note (♩.) lasting for 1.5 second, the dotted eighth note (♪) 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.

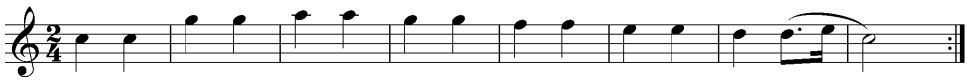
2.8 Musical Variations

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.

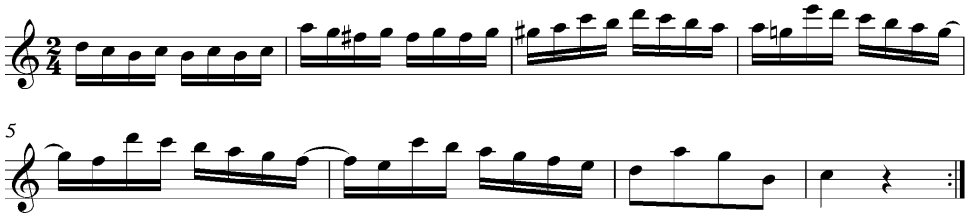
Changes in variation may affect multiple musical elements, including melody, rhythm, harmony, counterpoint, timbre, and orchestration. A variation can involve modifying a single musical component or multiple elements simultaneously. For example, a composer might alter the melody by adding embellishments 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.

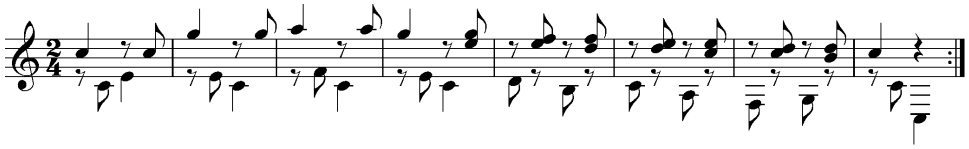
Melodic and rhythmic variations play essential roles in musical development and artistic expression. Composers often use these techniques to introduce contrast, maintain listener interest, and develop thematic material throughout a piece. By applying variation techniques, musicians can generate new perspectives 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.



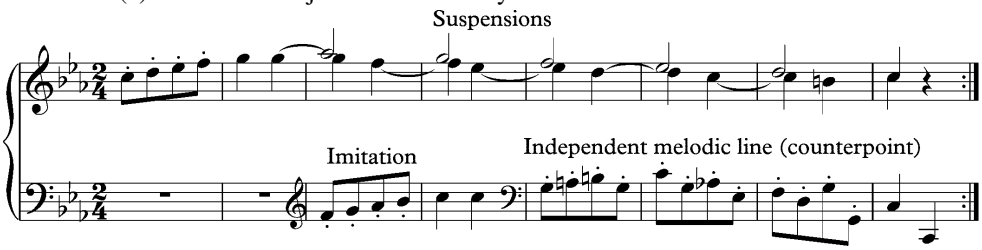
(a) The original of Ah vous dirai-je Maman in the first 8 bars.



(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.



(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

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

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

Given a sequence $\{\phi_i(kh)\}_{k=0}^{m-1}$, where $\phi_i : h\mathbb{N}_{m-1} \to \mathbb{R}$ 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

$$\alpha(\phi_i(kh)) := p_k \tag{3.2}$$

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:

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

where $\operatorname{dom}(\phi_i; m, h) = \{t \in \operatorname{dom}\phi_i : t \leq (m-1)h\}$. 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

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

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

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

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

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

To illustrate this mapping, consider the following cases:

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

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

   and since there is no $c \in \mathrm{dom}(\phi_1; 11, 0.01)$ 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 $\left\{\tilde{\phi}_1(kh)\right\}_{k=0}^{10}$ to new musical pitches.

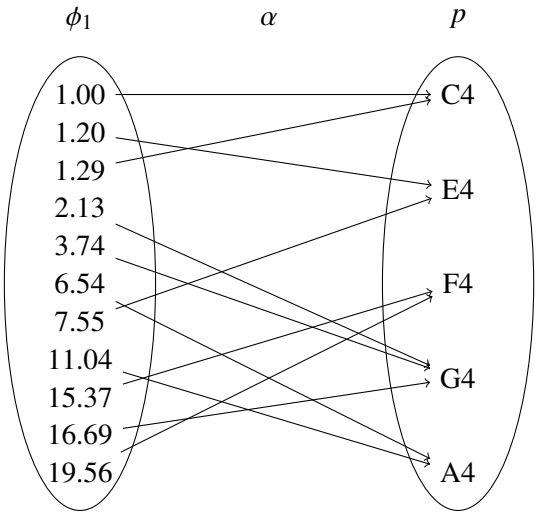


Figure 3.1: Mapping α.

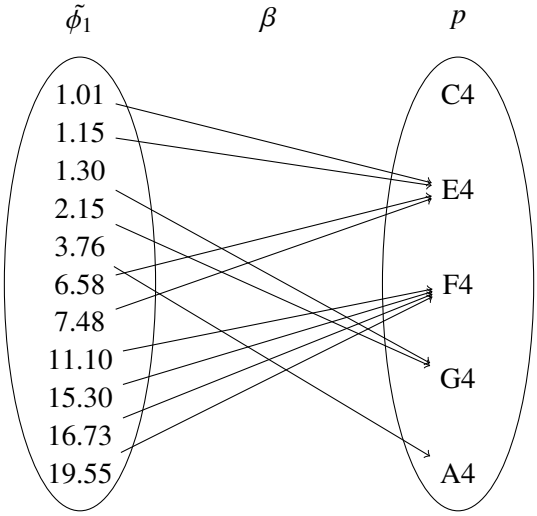
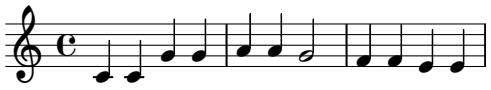
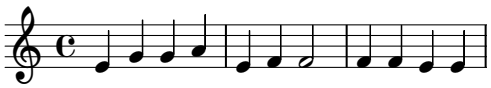


Figure 3.2: Mapping β.

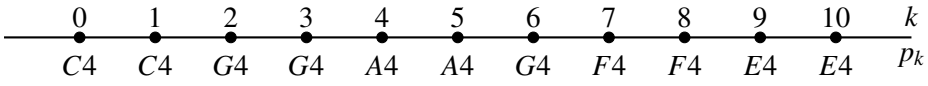


(a) The original of Ah vous dirai-je Maman in the first 3 bars.

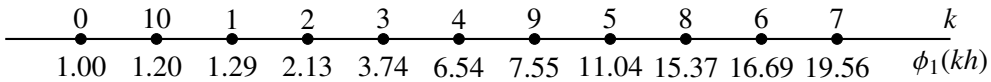


(b) The new variation of Ah vous dirai-je Maman in the first 3 bars, generated by musical variations from a chaotic mapping method with the initial condition $(1.01, 1, 1)$.

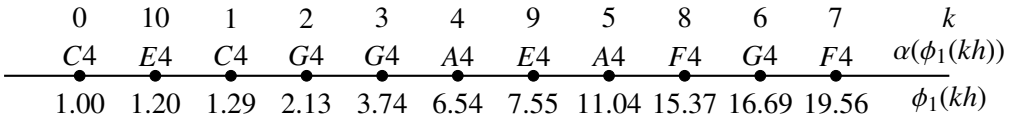
Figure 3.3: The original and new variation of Ah vous dirai-je Maman.



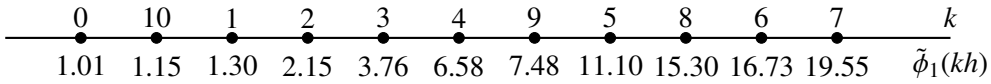
(a) The first 11 pitches of the Ah vous dirai-je Maman are marked below the 1D axis.



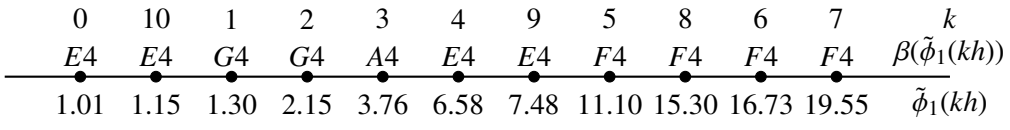
(b) The first 11 numerical solution in first component to the system (3.4) with initial condition of $(1, 1, 1)$ are marked below the 1D axis.



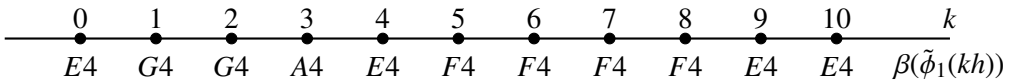
(c) For each component in $\{\phi_1(kh)\}_{k=0}^{10}$, apply the $\alpha(\phi_1(kh))$ mapping.



(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 $\vec{\beta}(\tilde{\phi}_1(kh))$ 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.

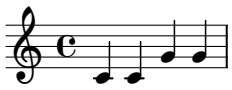
3.2 Expanded Rhythm Method

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 $p_k = p'_{kq} = p'_{kq+1} = \cdots = p'_{q(k+1)-1}$.

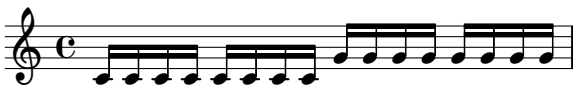
Example 3.2. The sequence $\{p_k\}_{k=0}^{3} = \{C4, C4, G4, G4\}$ represents the sequence of musical pitches in the first bar of the music piece "Ah vous dirai-je Maman," as illustrated in Figure 3.5a. Assuming $q = 4$ denotes the number of expanded notes for each p_k, we can establish an expanded sequence 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 $f : \mathbb{R}_+ \times \mathbb{R}^n \to \mathbb{R}^n$ is a continuous function.

Given a sequence $\{\phi_i(kh)\}_{k=0}^{mq-1}$, where $\phi_i : h\mathbb{N}_{mq-1} \to \mathbb{R}$ 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 $\overrightarrow{\beta}$ as:

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

where $\operatorname{dom}(\phi_i; mq, h) = \{t \in \operatorname{dom}\phi_i : t \leq (mq-1)h\}$. Resulting in the sequence $\{\overrightarrow{\beta}(\tilde{\phi}_i(kh))\}_{k=0}^{mq-1}$, 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:

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

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

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

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

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:

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

Similarly, we may use the sequence $\{\overleftarrow{\beta}(\tilde{\phi}_i(kh))\}_{k=0}^{mq-1}$ instead of $\{\overrightarrow{\beta}(\tilde{\phi}_i(kh))\}_{k=0}^{mq-1}$ 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

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

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

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

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

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

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

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

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

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

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

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:

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

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

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

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 $\{\vec{\beta}(\tilde{\phi}_1(kh))\}_{k=0}^{13}$, 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
$\vec{\beta}(\tilde{\phi}_1(kh))$	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 $\left\{\tilde{\phi}_1(kh)\right\}_{k=0}^{13}$ to new expanded musical pitches.

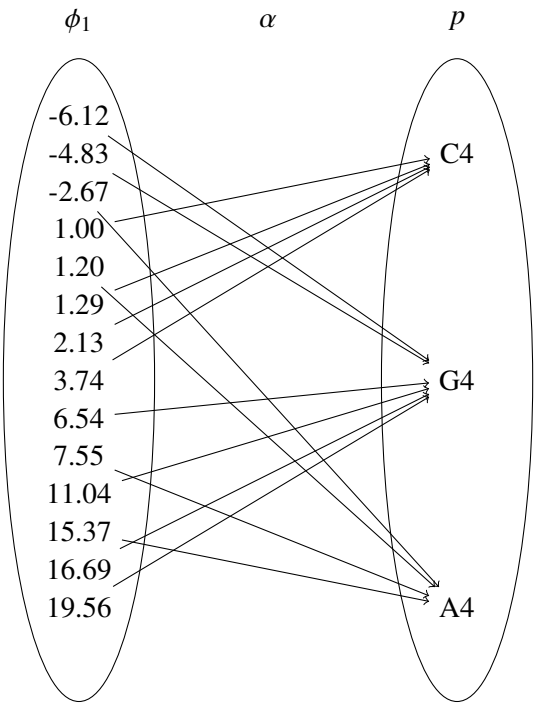


Figure 3.6: Mapping process of α

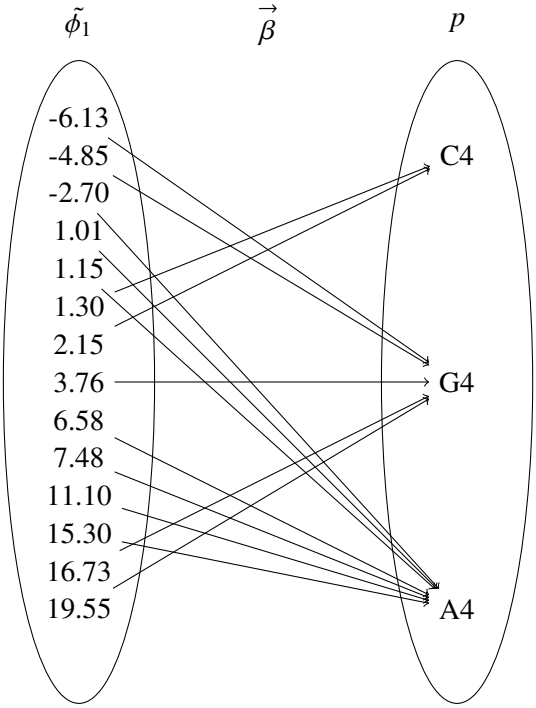
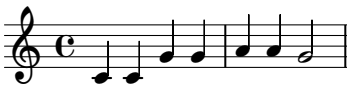
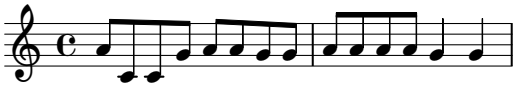


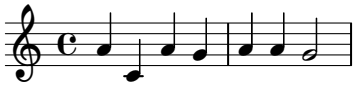
Figure 3.7: Mapping process of $\vec{\beta}$



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

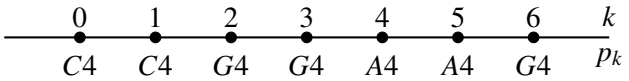


(b) The new expanded rhythm variation of Ah vous dirai-je, maman in the first 2 bar.



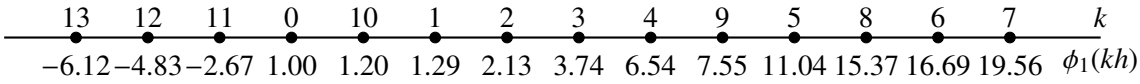
(c) The new variation of Ah vous dirai-je, maman in the first 2 bar.

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

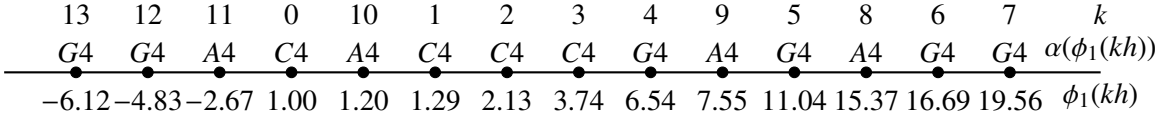


(a) The first 7 pitches of the Ah vous dirai-je Maman are marked below the 1D axis.

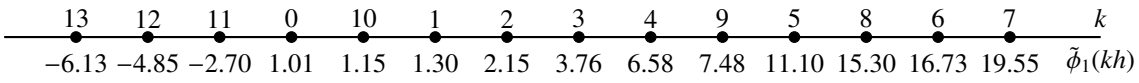
(b) The first 14 expanded rhythm pitches of the Ah vous dirai-je Maman are marked below the 1D axis.



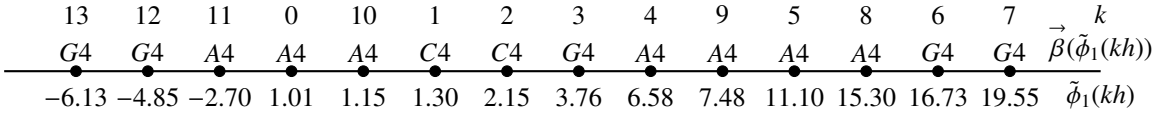
(c) The first 14 numerical solution in first component to the system (3.9) with initial condition of $(1, 1, 1)$ are marked below the 1D axis.



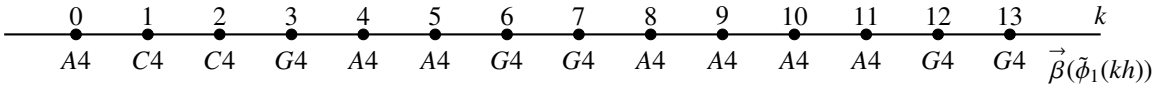
(d) For each component in $\{\phi_1(kh)\}_{k=0}^{6}$, apply the $\alpha(\phi_1(kh))$ mapping.



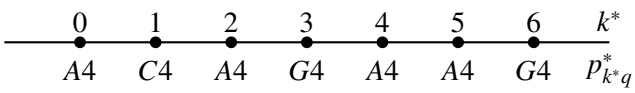
(e) The first 14 numerical solution in first component to the system (3.9) with new initial condition of $(1.01, 1, 1)$ are marked below the 1D axis.



(f) For each component in $\{\tilde{\phi}_1(kh)\}_{k=0}^{13}$, apply the $\vec{\beta}(\tilde{\phi}_1(kh))$ 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 3.9: 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 $\overleftarrow{\beta}$ from $\left\{\tilde{\phi}_1(kh)\right\}_{k=0}^{13}$ to new musical pitches according to the mapping (3.8), we obtain the sequence:

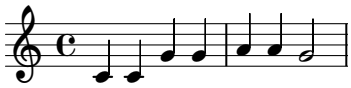
$$\{\overleftarrow{\beta}(\tilde{\phi}_1(kh))\}_{k=0}^{13} = \{C4, C4, C4, C4, G4, G4, A4, G4, G4, G4, C4, G4, G4, G4\},$$

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 $\{\overrightarrow{\beta}(\tilde{\phi}_1(kh))\}_{k=0}^{13}$, the sequence of new musical pitches is obtained as follows:

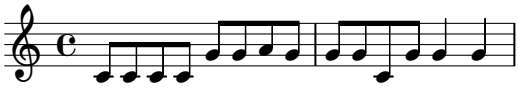
$$\{p^*_{k^*q}\}_{k^*=0}^{6} = \{A4, C4, A4, G4, A4, A4, G4\},$$

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

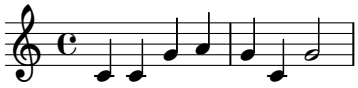
This example shows that changing the mapping from $\overrightarrow{\beta}$ to $\overleftarrow{\beta}$ can create different pitches for new variations. However, determining the best mapping for generating new variations depends on the listener's preference. We provide $\overleftarrow{\beta}$ as an alternative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



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

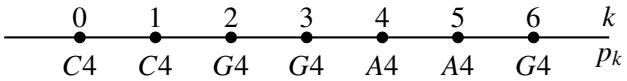


(b) The new expanded rhythm variation of Ah vous dirai-je, maman in the first 2 bar.



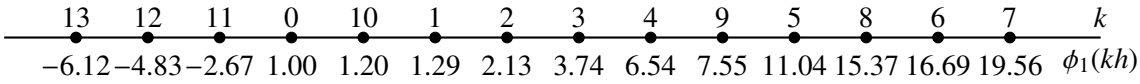
(c) The new variation of Ah vous dirai-je, maman in the first 2 bar.

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

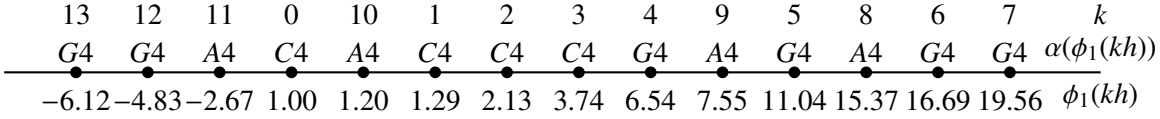


(a) The first 7 pitches of the Ah vous dirai-je Maman are marked below the 1D axis.

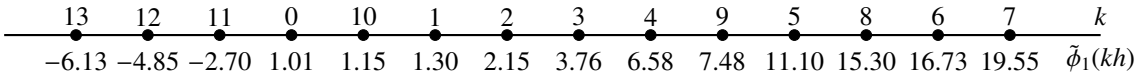
(b) The first 14 expanded rhythm pitches of the Ah vous dirai-je Maman are marked below the 1D axis.



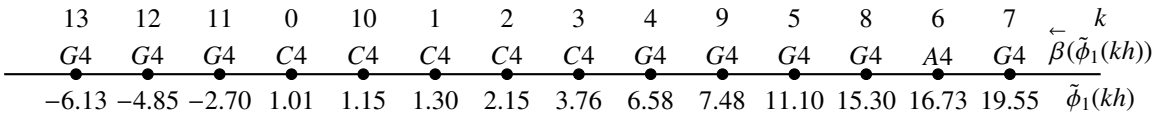
(c) The first 14 numerical solution in first component to the system (3.9) with initial condition of $(1, 1, 1)$ are marked below the 1D axis.



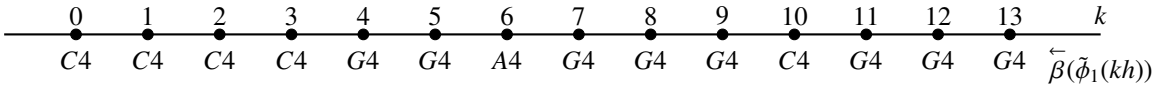
(d) For each component in $\{\phi_1(kh)\}_{k=0}^{6}$, apply the $\alpha(\phi_1(kh))$ mapping.



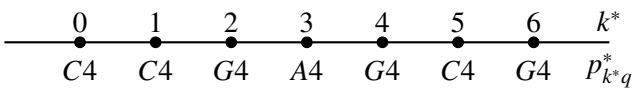
(e) The first 14 numerical solution in first component to the system (3.9) with new initial condition of $(1.01, 1, 1)$ are marked below the 1D axis.



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

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 ($\mathfrak{G}$). Furthermore, let the sequence $\{p'_k\}$ represent the rhythmically expanded pitches, where each p_k is subdivided into four musical notes. Let

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

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 $\overrightarrow{\beta}$, as defined in (3.7), the resulting variation is depicted in Figure 3.13 within the Treble Clef ($\mathfrak{G}$). Conversely, when employing the expanded rhythm method with the mapping $\overleftarrow{\beta}$, as defined in (3.8), the corresponding variation is illustrated in Figure 3.14, also within the Treble Clef ($\mathfrak{G}$).

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 ($\mathfrak{F}$). Furthermore, let the sequence $\{p'_k\}$ represent the rhythmically expanded pitches, where each p_k is subdivided into two musical notes. Let

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

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 $\overrightarrow{\beta}$, as defined in (3.7), the resulting variation appears in Figure 3.13 in the Bass Clef ($\mathfrak{F}$). Conversely, applying the mapping $\overleftarrow{\beta}$, as defined in (3.8), produces the variation depicted in Figure 3.14 in the Bass Clef ($\mathfrak{F}$).

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

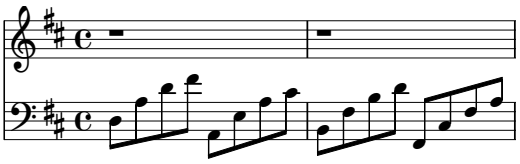


Figure 3.12: First two bars of Pachelbel's Canon.

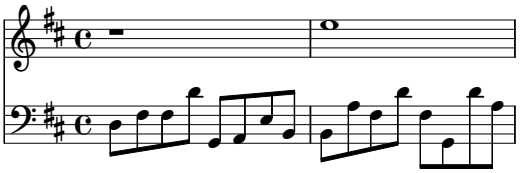


Figure 3.13: New variation of Pachelbel's Canon using the mapping $\overrightarrow{\beta}$ in the first two bars.

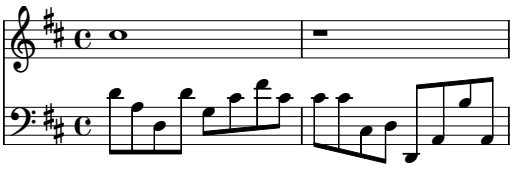


Figure 3.14: New variation of Pachelbel's Canon using the mapping $\overleftarrow{\beta}$ in the first two bars.

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 ($\mathfrak{G}$). Furthermore, let the sequence $\{p'_k\}$ represent the rhythmically expanded pitches, where each p_k is subdivided into four musical notes. Let

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

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 $\overrightarrow{\beta}$, as defined in (3.7), the resulting variation is depicted in Figure 3.16 within the Treble Clef ($\mathfrak{G}$). Conversely, when employing the expanded rhythm method with the mapping $\overleftarrow{\beta}$, as defined in (3.8), the corresponding variation is illustrated in Figure 3.17, also within the Treble Clef ($\mathfrak{G}$).

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 ($\mathfrak{F}$). Furthermore, let the sequence

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

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

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 $\overrightarrow{\beta}$, as defined in (3.7), the resulting variation appears in Figure 3.16 in the Bass Clef ($\mathfrak{Y}$). Conversely, applying the mapping $\overleftarrow{\beta}$, as defined in (3.8), produces the variation depicted in Figure 3.17 in the Bass Clef ($\mathfrak{Y}$).

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

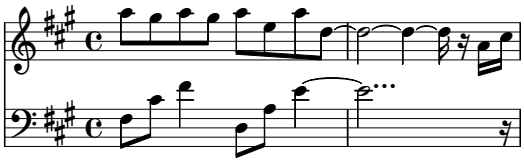


Figure 3.15: First two bars of River Flows In You.

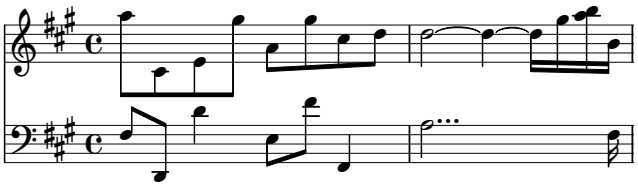


Figure 3.16: New variation of River Flows In You using the mapping $\overrightarrow{\beta}$ in the first two bars.

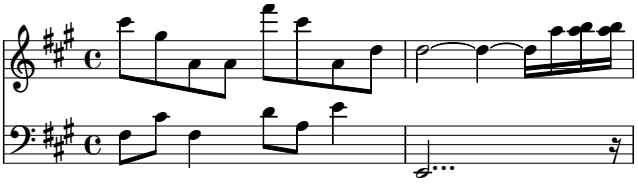


Figure 3.17: New variation of River Flows In You using the mapping $\overleftarrow{\beta}$ in the first two bars.

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 $\overleftarrow{\beta}$ 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 $\overrightarrow{\beta}$ 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:

```python
from melody import Melody
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:

```python
def lorenz_system(t, x, sigma=10, rho=28, beta=8/3):
    """
    Computes the derivatives of the Lorenz system at a given time t.
    Parameters:
    t (int or float): Time variable (not explicitly used in computation).
    x (array_like): State vector at time t, given as a list or NumPy array
    ↪ [x_1, x_2, x_3].
    sigma (float): Lorenz system parameter (must be greater than 0).
    rho (float): Lorenz system parameter (must be greater than 0).
    beta (float): Lorenz system parameter (must be greater than 0).

    Returns:
    numpy.ndarray: The new state vector [x_1dot, x_2dot, x_3dot] as a NumPy
    ↪ array.
    """
```

```
    x_1, x_2, x_3 = x
    x_1dot = sigma * (x_2 - x_1)
    x_2dot = x_1 * (rho - x_3) - x_2
    x_3dot = x_1 * x_2 - beta * x_3
    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:

```
import numpy as np
from melody import Melody

atb = Melody()
atb.load("Canon_in_D.mid")
atb.fit(np.array([1.5, 1.5, 1.5]))
atb.variate(np.array([1.51, 1.51, 1.51]))
atb.export("midi")
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:

```
def rossler_system(t, x, a, b, c):
    """
    Computes the derivatives of the Rössler system at a given time t.

    Parameters:
    t (int or float): Time variable (not explicitly used in computation).
    x (array_like): State vector at time t, given as a list or NumPy array
     ↪  [x_1, x_2, x_3].
    a (float): Rössler system parameter (must be greater than 0).
    b (float): Rössler system parameter (must be greater than 0).
    c (float): Rössler system parameter (must be greater than 0).

    Returns:
    numpy.ndarray: The new state vector [x_1dot, x_2dot, x_3dot] as a NumPy
     ↪  array.
    """
    x_1, x_2, x_3 = x
    x_1dot = -x_2 - x_3
    x_2dot = x_1 + a * x_2
    x_3dot = b + x_3 * (x_1 - c)
    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:

```
import numpy as np
from melody import Melody

def rossler_system(t, x, a, b, c):
    """
    Computes the derivatives of the Rössler system at a given time t.

    Parameters:
    t (int or float): Time variable (not explicitly used in computation).
    x (array_like): State vector at time t, given as a list or NumPy array
     ↪  [x_1, x_2, x_3].
    a (float): Rössler system parameter (must be greater than 0).
    b (float): Rössler system parameter (must be greater than 0).
    c (float): Rössler system parameter (must be greater than 0).

    Returns:
    numpy.ndarray: The new state vector [x_1dot, x_2dot, x_3dot] as a NumPy
     ↪  array.
    """
    x_1, x_2, x_3 = x
    x_1dot = -x_2 - x_3
    x_2dot = x_1 + a * x_2
    x_3dot = b + x_3 * (x_1 - c)
    return np.array([x_1dot, x_2dot, x_3dot])

atb = Melody()
atb.load("Canon_in_D.mid")
atb.set_dynamic(rossler_system, a=0.2, b=0.2, c=5.7)
atb.set_dynamic_sequence(1)
atb.fit(np.array([1.5,1.5,1.5]), method="expand", divisions=4)
atb.variate(np.array([1.51, 1.51, 1.51]), track=['Piano_0'], add_note=50)
atb.export("midi")
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:

```
def __init__(self):
        self.tracks = {}
        self.maps = {}
        self.dynamic = self.lorenz_system

        # Main Instance Attributes
        self.original_midi_data = None
        self.new_midi_data = None

        # Additional Instance Attributes
        self.newtracks = {}
        self.main_stream = None
        self.new_stream = None
        self.duration = {}
        self.new_duration = {}
        self.original_initial_condition = None
        self.new_initial_condition = None
        self.dynamic_sequence = 0
        self.notetrack = {}
        self.tracks_index = {}
        self.dynamic_parameter = {}

        # Support Instance Attributes
        self.method = None
        self.divisions = None
        self.backtrack_expand_index = []

        # Path
        self.lilypond_path = str(pathlib.Path("__file__").parent.resolve()) +
        ↪  "\\lilypond-2.24.4\\bin\\lilypond.exe"

        # etc.
        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:

```
import numpy as np
from melody import Melody
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:

```
>>> import numpy as np
>>> from melody import Melody
>>> atb = Melody()
>>> 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:

```
>>> import numpy as np
>>> from melody import Melody
>>> atb = Melody()
>>> 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:

```
def rossler_system(t, x, a, b, c):
    x_1, x_2, x_3 = x
    x_1dot = -x_2 - x_3
    x_2dot = x_1 + a * x_2
    x_3dot = b + x_3 * (x_1 - c)
    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:

```
>>> import numpy as np
>>> from melody import Melody
>>> atb = Melody()
>>> 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:

```
def rossler_system(t, x, a, b, c):
    x_1, x_2, x_3 = x
    x_1dot = -x_2 - x_3
    x_2dot = x_1 + a * x_2
    x_3dot = b + x_3 * (x_1 - c)
    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:

```
>>> import numpy as np
>>> from melody import Melody
>>> atb = Melody()
>>> atb.set_dynamic(lorenz_system)
>>> 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:

```
>>> import numpy as np
>>> from melody import Melody
>>> atb = Melody()
>>> 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:

```
>>> import numpy as np
>>> from melody import Melody
>>> atb = Melody()
>>> atb.load("Canon_in_D.mid")
>>> 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:

```
>>> import numpy as np
>>> from melody import Melody
>>> atb = Melody()
>>> atb.load("Canon_in_D.mid")
>>> 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:

```
>>> import numpy as np
>>> from melody import Melody
>>> atb = Melody()
>>> atb.load("Canon_in_D.mid")
>>> atb.fit(np.array([1.5,1.5,1.5]))
>>> atb.show_all_track()
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 $\overrightarrow{\beta}$ or $\overleftarrow{\beta}$ 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 $\overrightarrow{\beta}$ and `"left"` for mapping $\overleftarrow{\beta}$.

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:

```
>>> import numpy as np
>>> from melody import Melody
>>> atb = Melody()
>>> atb.load("Canon_in_D.mid")
>>> atb.fit(np.array([1.5, 1.5, 1.5]), method="expand", divisions=2)
>>> atb.show_all_track()
dict_keys(['Piano_0', 'track_1'])
>>> 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:

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

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

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

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

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

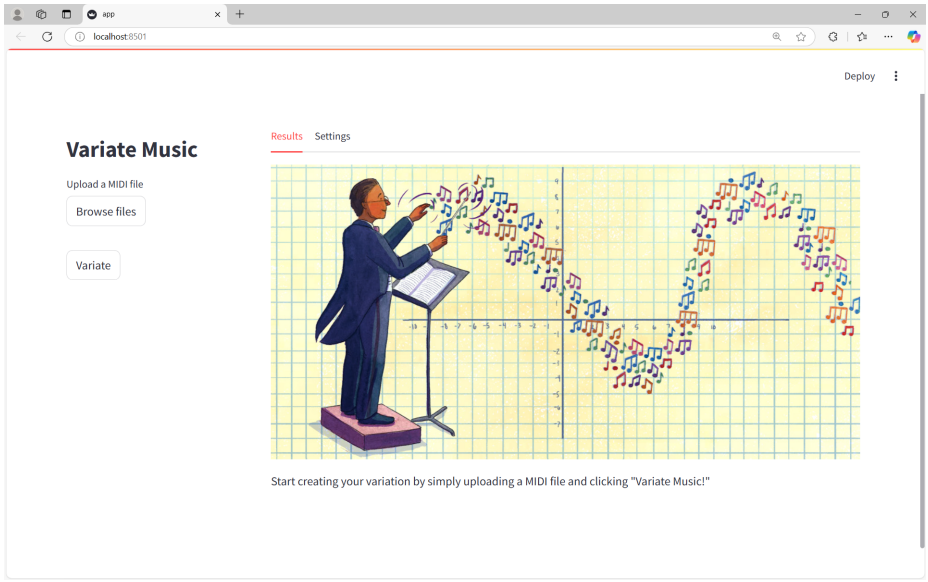
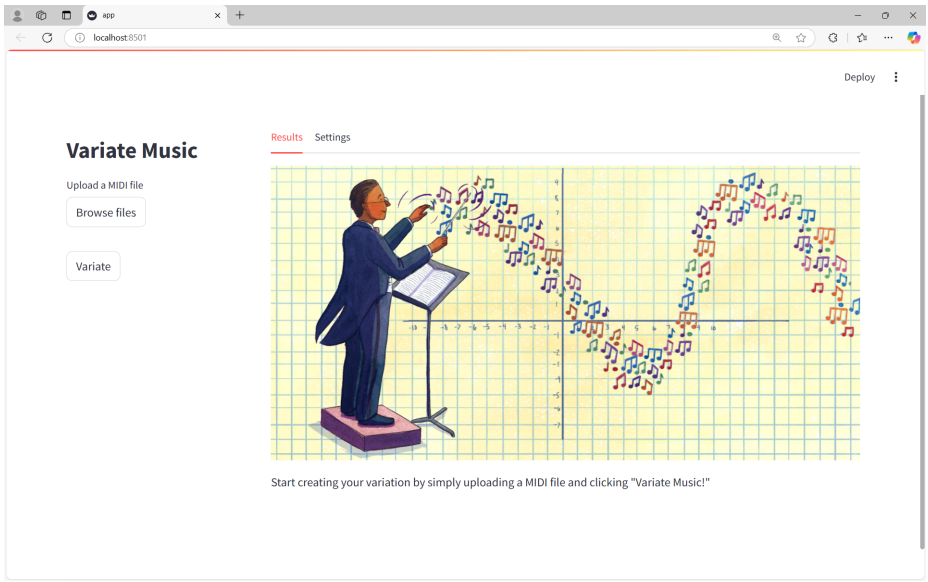


Figure 4.1: Web application interface.

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.

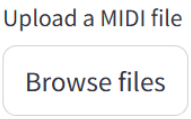
2) Instructions and Example

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:

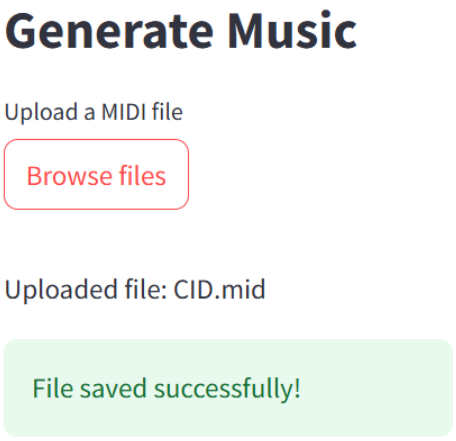


To generate a musical variation, follow these two steps:

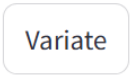
Step 1: Click the `Browse files` button and select the MIDI file you want to use for generating a musical variation.



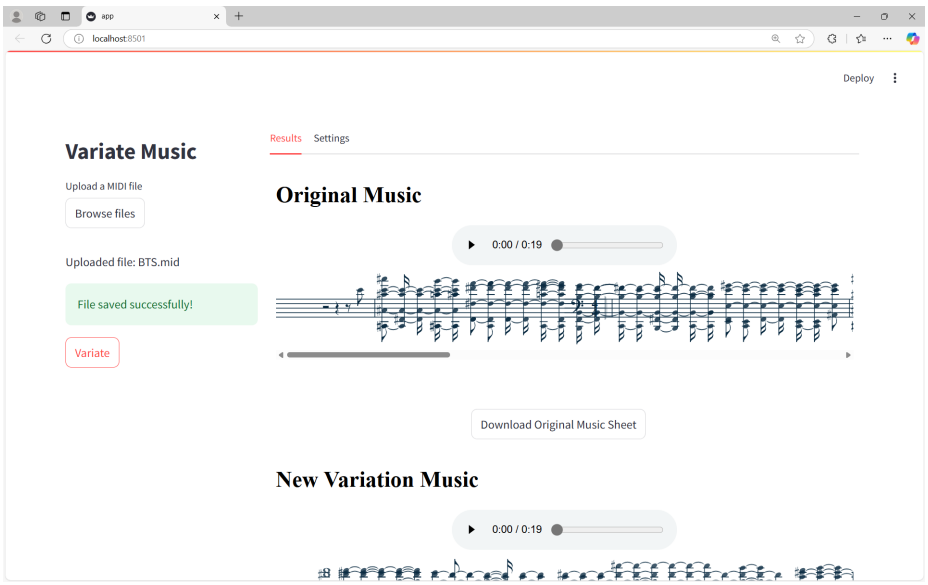
If your file has the `.mid` extension, the website will display the message: **File saved successfully!**



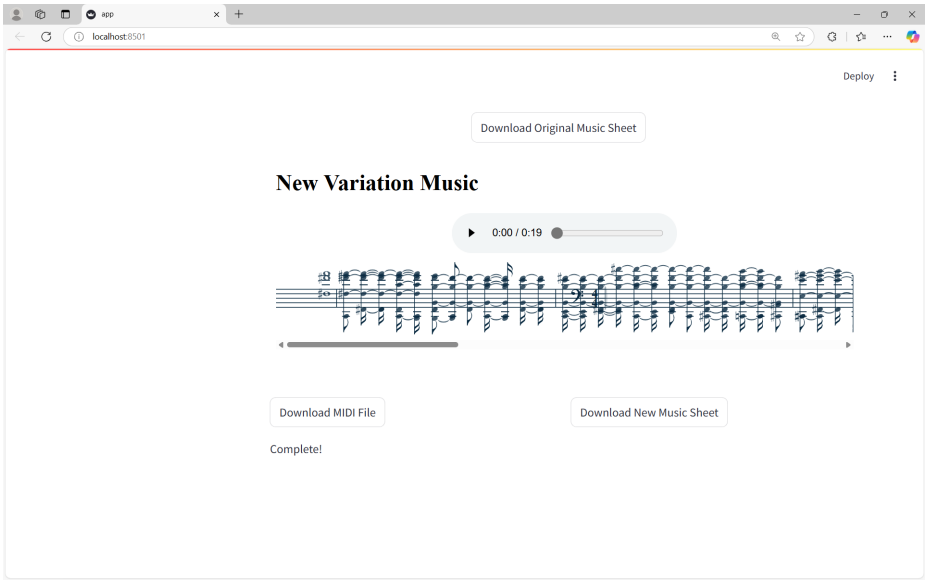
Step 2: Click the `Vairate` button to generate a musical variation from your imported MIDI file.



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



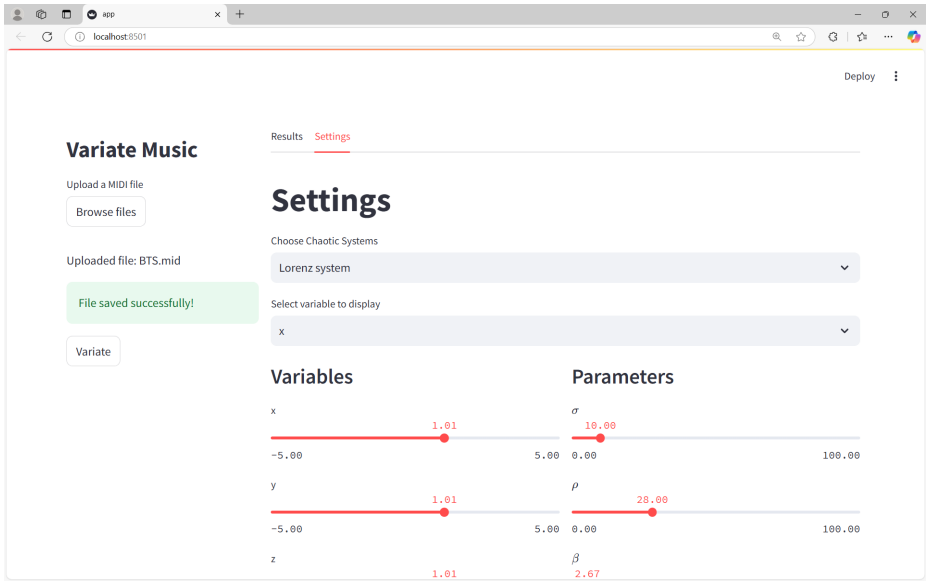
You can now listen to your new variation and download the MIDI file and music sheet by scrolling down on the website, where the following section will be displayed:



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:

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



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:

```
if 'x' not in st.session_state:
    st.session_state['x'] = 1.01
if 'y' not in st.session_state:
    st.session_state['y'] = 1.01
if 'z' not in st.session_state:
    st.session_state['z'] = 1.01
if 'sigma' not in st.session_state:
    st.session_state['sigma'] = 10.0
if 'rho' not in st.session_state:
    st.session_state['rho'] = 28.0
if 'beta' not in st.session_state:
    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:

```
if 'method' not in st.session_state:
    st.session_state['method'] = "Classic"
if 'option' not in st.session_state:
    st.session_state['option'] = 2
if 'sequence_select' not in st.session_state:
    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 componant in the chaotic system.

1.3) Initialization of File and Path Variables

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

```
if 'uploaded_file' not in st.session_state:
    st.session_state['uploaded_file'] = None
if 'save_path' not in st.session_state:
    st.session_state['save_path'] = None
if 'original_path' not in st.session_state:
    st.session_state['original_path'] = None
if 'new_path' not in st.session_state:
    st.session_state['new_path'] = None
```

```
if 'midi_new_download_path' not in st.session_state:
    st.session_state['midi_new_download_path'] = None
if 'pdf_original_path' not in st.session_state:
    st.session_state['pdf_original_path'] = None
if 'pdf_new_path' not in st.session_state:
    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:

```
if 'generate_clicked' not in st.session_state:
    st.session_state['generate_clicked'] = False
if 'complete_download' not in st.session_state:
    st.session_state['complete_download'] = False
if 'file_exist' not in st.session_state:
    st.session_state['file_exist'] = 0
if 'under_process' not in st.session_state:
    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:

```
if 'original_pdf_bytes' not in st.session_state:
    st.session_state['original_pdf_bytes'] = None
if 'new_midi_bytes' not in st.session_state:
    st.session_state['new_midi_bytes'] = None
if 'new_pdf_bytes' not in st.session_state:
    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:

```
if 'track_list' not in st.session_state:
    st.session_state['track_list'] = None
if 'backup_track_list' not in st.session_state:
    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:

```
if 'html_component' not in st.session_state:
    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:

```
if 'method_index' not in st.session_state:
    st.session_state['method_index'] = 0
if 'expanded_index' not in st.session_state:
    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:

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

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:

```
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:

```
css = '''
<style>
    [data-testid='stFileUploader'] {
        width: max-content;
    }
    [data-testid='stFileUploader'] section {
        padding: 0;
        float: left;
    }
    [data-testid='stFileUploader'] section > input + div {
        display: none;
    }
    [data-testid='stFileUploader'] section + div {
        display: none;
    }
</style>
'''
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:

```
if uploaded_file is not None:
    if st.session_state['file_exist'] == 1:
        print(f"now_file = {uploaded_file.name}")
        print(f"exist_file = {st.session_state['uploaded_file'].name}")
        if uploaded_file.name != st.session_state['uploaded_file'].name:
            st.session_state['track_list'] = None
            st.session_state['backup_track_list'] = None
    st.session_state['uploaded_file'] = uploaded_file
    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:

```
if 'uploaded_file' in st.session_state and st.session_state['uploaded_file']
↪  is not None:
    st.write("Uploaded file:", st.session_state['uploaded_file'].name)

    save_directory = os.path.join("static", "midi_file")
    os.makedirs(save_directory, exist_ok=True)

    if ' ' in str(st.session_state['uploaded_file'].name):
        st.warning('Warning: We can\'t save your MIDI file because your MIDI
        ↪  file name has spacing. Please fill the spacing with "_".')
    else:
        save_path = os.path.join(save_directory,
        ↪  st.session_state['uploaded_file'].name)

        # Reset file pointer to the beginning
        st.session_state['uploaded_file'].seek(0)

        # Save the uploaded file locally
        with open(save_path, "wb") as f:
            f.write(st.session_state['uploaded_file'].read())
        st.success("File saved successfully!")

        # Reset file pointer again to read for MIDI processing
        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:

```
try:
    # Load into your Melody object
    st.session_state['save_path'] = save_path
    my_melody.load(path=st.session_state['save_path'])
except Exception as e:
    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:

```
generate_botton = 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.

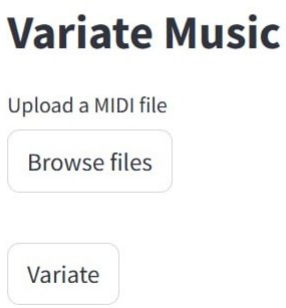


Figure 4.2: Graphical representation of the interface displaying the Sidebar.

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:

```
with right_col:
    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:

```
if generate_botton:
    st.session_state['generate_clicked'] = True
```

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

```
if st.session_state['generate_clicked'] == False:
    st.image("intro.jpg")
    st.write("Start creating your variation by simply uploading a MIDI file
    ↪  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:

```
if 'uploaded_file' in st.session_state and st.session_state['uploaded_file']
↪  is not None:
    if generate_botton == True:
        st.session_state['complete_download'] = False
        my_melody.set_dynamic_sequence(sequence=st.session_state['sequence_se
        ↪  lect'])
        key_list =
        ↪  my_melody.fit(original_initial_condition=[st.session_state['x'],
        ↪  st.session_state['y'], st.session_state['z']],
            method=st.session_state['method'],
            ↪  divisions=st.session_state['option'],
            dynamic=le, d=st.session_state['sigma'],
            ↪  r=st.session_state['rho'], b=st.session_state['beta'])
        if st.session_state['track_list'] == None:
            st.session_state['track_list'] = [key for key in key_list]
            st.session_state['backup_track_list'] = [key for key in key_list]
        my_melody.variate(new_initial_condition=[1.5, 1.5, 1.5],
        ↪  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:

```
original_path, new_path, midi_new_download_path = my_melody.export("midi")
st.session_state['original_path'] = original_path
st.session_state['new_path'] = new_path
st.session_state['midi_new_download_path'] = midi_new_download_path
with open(st.session_state['midi_new_download_path'], "rb") as file:
    st.session_state['new_midi_bytes'] = file.read()

pdf_original_path, pdf_new_path = my_melody.export("pdf")
st.session_state['pdf_original_path'] = pdf_original_path
with open(st.session_state['pdf_original_path'], "rb") as file:
    st.session_state['original_pdf_bytes'] = file.read()

st.session_state['pdf_new_path'] = pdf_new_path
with open(st.session_state['pdf_new_path'], "rb") as file:
    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:

```
components.html(
f"""
<h1 style="font-size:30px;">Original Music</h1>
<center>
<midi-player
  src="{st.session_state['original_path']}"
  sound-font visualizer="#myStaffVisualizer1">
</midi-player>

<midi-visualizer type="staff" id="myStaffVisualizer1"
  src="{st.session_state['original_path']}">
</midi-visualizer>
</center>
<script src="https://cdn.jsdelivr.net/combine/npm/tone@14.7.58,npm/@magenta/music@1.23.1/es6/core.js,npm/focus-visible@5,npm/html-midi-player@1.5.0"></script>
""",
height=300)
```

This block:

- Displays the original MIDI file with an interactive player and staff visualizer.
- Uses external JavaScript libraries for MIDI playback and visualization.

3.2.5) Download Buttons

Download buttons are provided for the original and new MIDI and PDF files:

```
original_col1, original_col2, original_col3, original_col4, original_col5 = st.columns([1,1,2,1,1])
with original_col3:
    st.download_button(
        label="Download Original Music Sheet",
        data=st.session_state['original_pdf_bytes'],
        file_name=str(st.session_state['uploaded_file'].name) + "_original.pdf",
        mime="application/pdf"
    )
```

This section:

- Organizes the download buttons into columns for better layout.
- Allows users to download the original and new MIDI and PDF files.

The output generated from the execution of all above implementation is illustrated in Figure 4.3, providing a visual representation of the expected results.

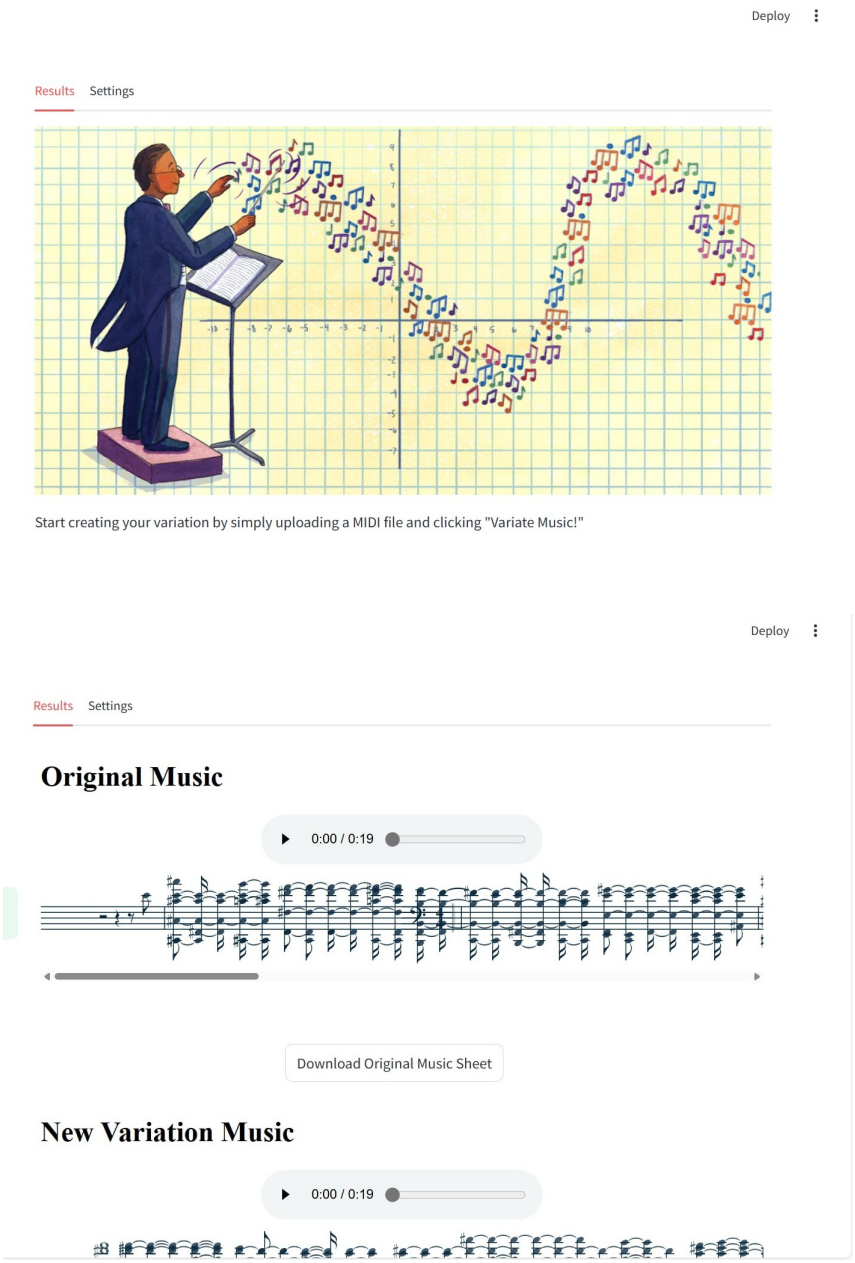


Figure 4.3: Graphical representation of the interface displaying the results webpage.

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:

```
equation = st.selectbox(
    "Choose Chaotic Systems",
    ("Lorenz system", "Chaotic system"),
    index=0,
    placeholder="Choose Equation"
)
```

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:

```
if equation == "Lorenz system":
    variable_Lorenz = st.selectbox(
        "Select variable to display",
        options=("x", "y", "z"),
        index=0
    )
    st.session_state['x'] = st.slider("x", min_value=-5.0, max_value=5.0, value=1.01, step=0.01)
    st.session_state['y'] = st.slider("y", min_value=-5.0, max_value=5.0, value=1.01, step=0.01)
    st.session_state['z'] = st.slider("z", min_value=-5.0, max_value=5.0, value=1.01, step=0.01)
    st.session_state['sigma'] = st.slider(r"$\sigma$", min_value=0.0, max_value=100.0, value=10.0, step=0.01)
    st.session_state['rho'] = st.slider(r"$\rho$", min_value=0.0, max_value=100.0, value=28.0, step=0.01)
    st.session_state['beta'] = st.slider(r"$\beta$", min_value=0.0, 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:

```
if st.button('Check Parameters'):
    my_melody.load(path=st.session_state['save_path'])
    my_melody.set_dynamic_sequence(sequence=st.session_state['sequence_select'])
    my_melody.fit(initial_condition=[st.session_state['x'],st.session_state['y'],st.session_state['z']], dynamic=le, d=st.session_state['sigma'], r=st.session_state['rho'], b=st.session_state['beta'])
    checker = my_melody.is_chaotic()
    if str(checker) == 'True':
        st.write(f"Chaotic parameter")
    else:
        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:

```
st.session_state['method'] = st.selectbox(
    "Additional Options",
    ("Classic", "Expanded"),
    index=st.session_state['method_index'],
    placeholder="Choose Equation",
)
```

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:

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

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.

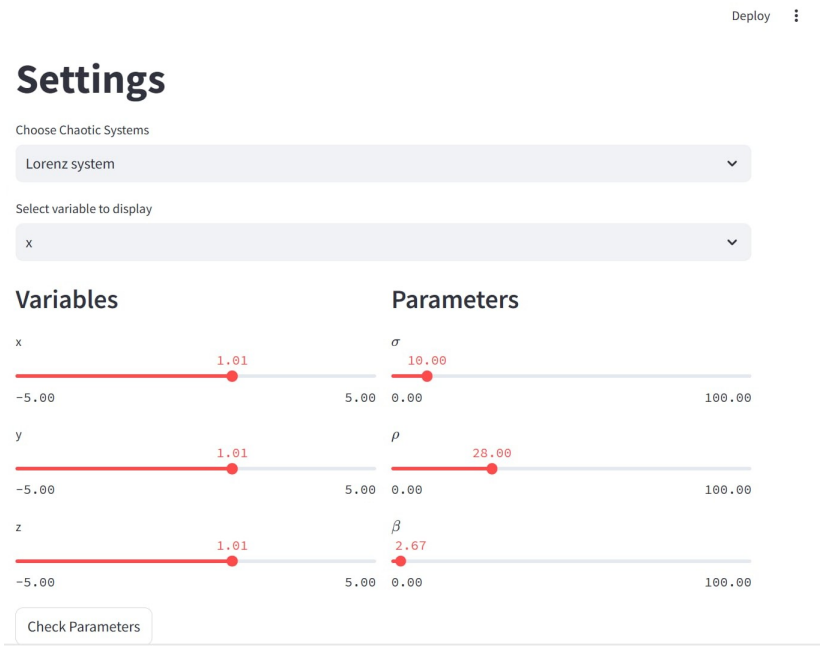


Figure 4.4: Graphical representation of the interface displaying the settings webpage.

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.

[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:

```
import music21
from music21 import stream, midi, scale, converter, instrument
from datetime import datetime as dt
import numpy as np
import subprocess
import os
import pathlib
import matplotlib.pyplot as plt

class Melody:

    def __init__(self):
        self.tracks = {}
        self.maps = {}
        self.dynamic = self.lorenz_system

        # Main Instance Attributes
        self.original_midi_data = None
        self.new_midi_data = None

        # Additional Instance Attributes
        self.newtracks = {}
        self.main_stream = None
        self.new_stream = None
        self.duration = {}
        self.new_duration = {}
        self.original_initial_condition = None
        self.new_initial_condition = None
        self.dynamic_sequence = 0
        self.notetrack = {}
        self.tracks_index = {}
        self.dynamic_parameter = {}

        # Support Instance Attributes
        self.method = None
        self.divisions = None
        self.backtrack_expand_index = []

        # Path
```

```
        self.lilypond_path =
        ↪  str(pathlib.Path("__file__").parent.resolve()) +
        ↪  "\\lilypond-2.24.4\\bin\\lilypond.exe"

        # etc.
        self.filename = None

    ##########################
    # Main Method
    ##########################

    def load(self, path):
        midi_file = midi.MidiFile()
        midi_file.open(path)
        midi_file.read()
        midi_file.close()

        self.original_midi_data =
        ↪  midi.translate.midiFileToStream(midi_file)
        self.new_midi_data = midi.translate.midiFileToStream(midi_file)

        self.main_stream = midi.translate.midiFileToStream(midi_file)
        self.new_stream = midi.translate.midiFileToStream(midi_file)
        self.filename = str(pathlib.Path(path).stem)

    def fit(self, original_initial_condition, method="classic",
    ↪  divisions=None):
        self.original_initial_condition = original_initial_condition
        self.method = method
        self.divisions = divisions

        if self.method == "classic":
            for index in range(len(self.main_stream)):
                instruments = self.main_stream[index][0].getElementsBy
                ↪  Class(instrument.Instrument)
                if instruments:
                    instrument_name =
                    ↪  str(instruments[0].instrumentName) + "_" +
                    ↪  str(index)
                else:
                    instrument_name = "track_" + str(index)
                music_list = []
                duration_list = []
                for part in self.main_stream[index]:
                    for n in part.flat.notesAndRests:
                        dur = n.duration.quarterLength
                        if n.isRest:
                            note_num = "<REST>"
                            music_list.append(note_num)
                            duration_list.append(dur)
                        else:
```

```
                            if len(n.pitches) == 1:
                                note_num = n.pitches[0].midi
                                music_list.append(note_num)
                                duration_list.append(dur)
                            else:
                                note_num = sorted(set([p.midi for p in
                                ↪ n.pitches]))
                                music_list.append(list(note_num))
                                duration_list.append(dur)
                    string_music_list = [element for element in music_list]
                    for i in range(len(music_list)):
                        try:
                            string_music_list[i] =
                            ↪ self.note_converter(music_list[i])
                        except TypeError:
                            for j in range(len(music_list[i])):
                                string_music_list[i][j] =
                                ↪ self.note_converter(music_list[i][j])

                    time, solution = self.rk4(time=[0,
                    ↪ 0.01*(len(music_list) - 1)], f=self.dynamic,
                    ↪ ini=self.original_initial_condition,
                    ↪ **self.dynamic_parameter)
                    main_sequence = solution[:, self.dynamic_sequence]
                    print(main_sequence[0:100:5])
                    #self.original_time = time
                    # self.original_trajectory = main_sequence

                    self.notetrack.update({instrument_name:string_music_li
                    ↪ st})
                    self.duration.update({instrument_name:duration_list})
                    self.tracks.update({instrument_name:main_sequence})
                    self.tracks_index.update({instrument_name:index})
                for key in self.tracks.keys():
                    dummy_list = []
                    for i in range(len(self.notetrack[key])):
                        dummy_list.append([self.tracks[key][i],
                        ↪ self.notetrack[key][i]])
                    self.maps.update({key:dummy_list})
            elif self.method == "expand":
                for index in range(len(self.main_stream)):
                    instruments = self.main_stream[index][0].getElementsBy
                    ↪ Class(instrument.Instrument)
                    if instruments:
                        instrument_name =
                        ↪ str(instruments[0].instrumentName) + "_" +
                        ↪ str(index)
                    else:
                        instrument_name = "track_" + str(index)
                    duration_list = []
                    for part in self.main_stream[index]:
```

```
                    for n in part.flat.notesAndRests:
                        dur = n.duration.quarterLength
                        if n.isRest:
                            duration_list.append(dur)
                        else:
                            if len(n.pitches) == 1:
                                duration_list.append(dur)
                            else:
                                duration_list.append(dur)
                self.duration.update({instrument_name:duration_list})
            for index in range(len(self.main_stream)):
                self.expand_note_durations(s=self.main_stream[index],
                ↪  divisions=self.divisions)
                instruments = self.main_stream[index][0].getElementsBy
                ↪  Class(instrument.Instrument)
                if instruments:
                    instrument_name =
                    ↪  str(instruments[0].instrumentName) + "_" +
                    ↪  str(index)
                else:
                    instrument_name = "track_" + str(index)
                music_list = []
                #duration_list = []
                for part in self.main_stream[index]:
                    for n in part.flat.notesAndRests:
                        dur = n.duration.quarterLength
                        if n.isRest:
                            note_num = "<REST>"
                            music_list.append(note_num)
                            #duration_list.append(dur)
                        else:
                            if len(n.pitches) == 1:
                                note_num = n.pitches[0].midi
                                music_list.append(note_num)
                                #duration_list.append(dur)
                            else:
                                note_num = sorted(set([p.midi for p in
                                ↪  n.pitches]))
                                music_list.append(list(note_num))
                                #duration_list.append(dur)
                string_music_list = [element for element in music_list]
                for i in range(len(music_list)):
                    try:
                        string_music_list[i] =
                        ↪  self.note_converter(music_list[i])
                    except TypeError:
                        for j in range(len(music_list[i])):
                            string_music_list[i][j] =
                            ↪  self.note_converter(music_list[i][j])
                self.backtrack_expand_index = [index for index in
                ↪  range(0, len(music_list) - 1, self.divisions)]
```

```
                time, solution = self.rk4(time=[0,
                ↪  0.01*(len(music_list) - 1)], f=self.dynamic,
                ↪  ini=self.original_initial_condition,
                ↪  **self.dynamic_parameter)
                main_sequence = solution[:, self.dynamic_sequence]
                #self.original_time = time
                # self.original_trajectory = main_sequence

                self.notetrack.update({instrument_name:string_music_li」
                ↪  st})
                self.tracks.update({instrument_name:main_sequence})
                self.tracks_index.update({instrument_name:index})
            for key in self.tracks.keys():
                dummy_list = []
                for i in range(len(self.notetrack[key])):
                    dummy_list.append([self.tracks[key][i],
                    ↪  self.notetrack[key][i]])
                self.maps.update({key:dummy_list})
        else:
            raise Exception("Out of Option!")
        #return self.maps
        # return self.tracks.keys()
        #return print(self.tracks.keys())

    def variate(self, new_initial_condition, track=None,
    ↪  criteria="right", add_note=0):
        self.new_initial_condition = new_initial_condition
        if track == None:
            track = list(self.tracks.keys())
        if add_note != 0:
            for key in track:
                for each_note in range(add_note):
                    # Access parts from the stream
                    parts = self.new_stream.getElementsByClass('Part')
                    target_part = parts[self.tracks_index[key]]

                    # Determine the last measure in the part
                    if target_part.measure(-1):
                        last_measure = target_part.measure(-1)  # Get
                        ↪  the last measure
                    else:
                        # If no measures exist, create the first one
                        last_measure = music21.stream.Measure()
                        target_part.append(last_measure)

                    # Create the new note
                    new_note = music21.note.Note('C4')
                    duration_len = len(self.duration[key])
                    new_note.quarterLength =
                    ↪  self.duration[key][each_note % duration_len]

```

```
                    # Append the new note to the last measure
                    last_measure.append(new_note)
                #print(target_part[-1].show("text"))
                #print("-"*100)

        if self.method == "expand":
            for index in range(len(self.new_stream)):
                self.expand_note_durations(s=self.new_stream[index],
                ↪  divisions=self.divisions)
                instruments = self.new_stream[index][0].getElementsByC
                ↪  lass(instrument.Instrument)
                if instruments:
                    instrument_name =
                    ↪  str(instruments[0].instrumentName) + "_" +
                    ↪  str(index)
                else:
                    instrument_name = "track_" + str(index)
                duration_list = []
                for part in self.new_stream[index]:
                    for n in part.flat.notesAndRests:
                        dur = n.duration.quarterLength
                        duration_list.append(dur)
                self.new_duration.update({instrument_name:duration_lis
                ↪  t})
            #self.duration = self.new_duration

        for key in track:
            sorted_tracks = sorted(self.maps[key], key=lambda x: x[0])
            if self.method == "classic":
                dummy_list = [None]*(len(self.tracks[key]) + add_note)
                time, solution = self.rk4(time=[0,
                ↪  0.01*(len(dummy_list) - 1)], f=self.dynamic,
                ↪  ini=self.new_initial_condition,
                ↪  **self.dynamic_parameter)
                main_sequence = solution[:, self.dynamic_sequence]
            elif self.method == "expand":
                dummy_list = [None]*(len(self.tracks[key]) +
                ↪  add_note)*self.divisions
                time, solution = self.rk4(time=[0,
                ↪  0.01*(len(dummy_list) - 1)], f=self.dynamic,
                ↪  ini=self.new_initial_condition,
                ↪  **self.dynamic_parameter)
                main_sequence = solution[:, self.dynamic_sequence]
                # self.new_time = time
                # self.new_trajectory = main_sequence
            else:
                raise Exception("Out of Option!")

            if criteria == "right":
                for i in range(len(main_sequence) if len(main_sequence)
                ↪  == len(dummy_list) else len(main_sequence) - 1):
```

```
                    for j in range(len(sorted_tracks)):
                        if main_sequence[i] <= sorted_tracks[j][0]:
                            dummy_list[i] = sorted_tracks[j][1]
                            break
                        elif main_sequence[i] > sorted_tracks[-1][0]:
                            dummy_list[i] = sorted_tracks[-1][1]
                            break
            elif criteria == "left":
                for i in range(len(main_sequence) if len(main_sequence)
                ↪  == len(dummy_list) else len(main_sequence) - 1):
                    for j in range(len(sorted_tracks)):
                        if main_sequence[i] <= sorted_tracks[j][0]:
                                dummy_list[i] = sorted_tracks[j - 1][1]
                                break
                        elif main_sequence[i] > sorted_tracks[-1][0]:
                            dummy_list[i] = sorted_tracks[-1][1]
                            break
                        elif (main_sequence[i] <= sorted_tracks[0][0])
                         ↪  and (j - 1 < 0):
                            main_sequence[i] = sorted_tracks[0][0]
                            break
            else:
                raise Exception("Out of Option!")

            self.newtracks.update({key:dummy_list})
        #return self.newtracks

    def export(self, format_type):
        exist_track = [track for track in self.newtracks.keys()]
        exist_new_stream = self.new_midi_data
        if self.method == "classic":
            for key in self.tracks.keys():
                if key in exist_track:
                    index = 0
                    for element in exist_new_stream[self.tracks_index[」
                     ↪  key]].recurse():
                        if isinstance(element, (music21.note.Note,
                         ↪  music21.chord.Chord, music21.note.Rest)):
                            replacement = self.newtracks[key][index]
                            duration_value = self.duration[key][index]
                            if replacement == '<REST>':
                                new_element = music21.note.Rest()
                            elif isinstance(replacement, list):
                                new_element =
                                 ↪  music21.chord.Chord(replacement)
                            else:
                                new_element =
                                 ↪  music21.note.Note(replacement)
                            new_element.duration = music21.duration.Du」
                             ↪  ration(duration_value)
```

```
                        element.activeSite.replace(element,
                        ↪ new_element)
                        #print(f"{element} --> {replacement}")
                        if index < len(self.newtracks[key]) - 1:
                            index += 1
                        else:
                            break
        elif self.method == "expand":
            for key in self.tracks.keys():
                if key in exist_track:
                    index = 0
                    for element in exist_new_stream[self.tracks_index[」
                    ↪ key]].recurse():
                        if isinstance(element, (music21.note.Note,
                        ↪ music21.chord.Chord, music21.note.Rest)):
                            replacement = self.newtracks[key][self.bac」
                            ↪ ktrack_expand_index[index]]
                            duration_value = self.duration[key][index]
                            if replacement == '<REST>':
                                new_element = music21.note.Rest()
                            elif isinstance(replacement, list):
                                new_element =
                                ↪ music21.chord.Chord(replacement)
                            else:
                                new_element =
                                ↪ music21.note.Note(replacement)
                            new_element.duration = music21.duration.Du」
                            ↪ ration(duration_value)
                            element.activeSite.replace(element,
                            ↪ new_element)
                            #print(f"{element} --> {replacement}")
                            if index < len(self.backtrack_expand_index)
                            ↪ - 1:
                                index += 1
                            else:
                                break
                #print("-"*50)
        #print(self.new_stream.show("text"))
        current_path = pathlib.Path("__file__").parent.resolve()
        date = str(dt.now().isoformat())[:-7]
        filepath = str(current_path) + "\\" + "file_store" + "\\" +
        ↪ self.filename #+ date.replace(":", "-")
        isExist = os.path.exists(filepath)
        if not isExist:
            os.makedirs(filepath)
        original = self.original_midi_data
        new = self.new_midi_data
        if format_type == "midi":
            # original.metadata = music21.metadata.Metadata()
            original.write('midi', filepath + "\\" + self.filename +
            ↪ '_original.mid')
```

```
            # new.metadata = music21.metadata.Metadata()
            new.write('midi', filepath + "\\" + self.filename +
             ↪  '_new.mid')
        elif format_type == "pdf":
            us = music21.environment.UserSettings()
            us['lilypondPath'] = self.lilypond_path

            original.metadata = music21.metadata.Metadata()
            original.metadata.title = self.filename.replace("_", " ")
             ↪  #+ " Original"
            conv = music21.converter.subConverters.ConverterLilypond()
            original_filepath = filepath + "\\" + self.filename +
             ↪  '_original'
            conv.write(original, fmt='lilypond', fp=original_filepath,
             ↪  subformats = ['pdf'])

            new.metadata = music21.metadata.Metadata()
            new.metadata.title = "Our Variation"
             ↪  #self.filename.replace("_", " ") + " New"
            new_conv =
             ↪  music21.converter.subConverters.ConverterLilypond()
            new_filepath = filepath + "\\" + self.filename + '_new'
            conv.write(new, fmt='lilypond', fp=new_filepath, subformats
             ↪  = ['pdf'])

    ##########################
    # Additional Method
    ##########################

    def show_all_track(self):
        return self.tracks.keys()

    def set_dynamic(self, dynamic, *args, **kwargs):
        self.dynamic = dynamic
        self.dynamic_parameter.update(*args, **kwargs)

    def set_dynamic_sequence(self, sequence):
        self.dynamic_sequence = sequence - 1

    # def plot_trajectory(self):
    #     plt.figure(figsize=(12,6), dpi=240)
    #     plt.xlabel("time")
    #     plt.ylabel("trajectory value at time")
    #     plt.plot(self.original_time, self.original_trajectory,
     ↪  label="Original Trajectory")
    #     plt.plot(self.new_time, self.new_trajectory, label="New
     ↪  Trajectory")
    #     plt.grid()
    #     plt.legend()
    #     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
            ↪ t.
381         Parameters:
382         t (int or float): Time variable (not explicitly used in
            ↪ computation).
383         x (array_like): State vector at time t, given as a list or
            ↪ NumPy array [x_1, x_2, x_3].
384         sigma (float): Lorenz system parameter (must be greater than
            ↪ 0).
385         rho (float): Lorenz system parameter (must be greater than 0).
386         beta (float): Lorenz system parameter (must be greater than 0).
387
388         Returns:
389         numpy.ndarray: The time derivatives [dx/dt, dy/dt, dz/dt] as a
            ↪ NumPy array.
390         """
391         x_1, x_2, x_3 = x
392         xdot = sigma * (x_2 - x_1)
393         ydot = x_1 * (rho - x_3) - x_2
394         zdot = x_1 * x_2 - beta * x_3
395         return np.array([xdot, ydot, zdot])
396
397     def rk4(self, time, ini, f, h=0.01, *args, **kwargs):
398         t = np.arange(time[0], time[1]+h, h)
399         row = len(t)
400         try:
401             var = len(ini)
402         except TypeError:
403             var = 1
404         x = np.zeros((row, var))
405         x[0, :] = ini
406         for j in range(0, len(t)-1):
407             k1 = f(t[j], x[j], *args, **kwargs)
408             k2 = f(t[j] + 0.5*h, x[j] +0.5*h*k1, *args, **kwargs)
409             k3 = f(t[j] + 0.5*h, x[j] + 0.5*h*k2, *args, **kwargs)
410             k4 = f(t[j] + h, x[j] + h*k3, *args, **kwargs)
411             x[j+1, :] = x[j] + (h/6)*(k1 + 2*k2 + 2*k3 + k4)
412         return t, x
413
414     def is_chaotic(self):
415         T = [0, 100]
416
417         # Initial state and parameters
418         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,
            ↪  rtol=1e-9, atol=1e-9)
422         time, sol = self.rk4(time=T, ini=initial_state, f=self.dynamic,
            ↪  **self.dynamic_parameter)
423
424         # Perturb the initial state slightly
425         perturbed_state = np.array([1.00001, 1.0, 1.0])
426
427         #perturbed_sol = solve_ivp(lorenz, t_span, perturbed_state,
            ↪  t_eval=t_eval, rtol=1e-9, atol=1e-9)
428         time, perturbed_sol = self.rk4(time=T, ini=perturbed_state,
            ↪  f=self.dynamic, **self.dynamic_parameter)
429
430         # Compute divergence between trajectories
431         divergence = np.linalg.norm(sol - perturbed_sol, axis=1)
432
433         # Estimate Lyapunov exponent
434         divergence_rate = np.log(divergence + 1e-10)  # Avoid log(0)
435         lyapunov_exponent = np.mean(divergence_rate[time > 10])  #
            ↪  Ignore transient phase
436
437         # Chaos detected if Lyapunov exponent is positive
438         return lyapunov_exponent > 0
439
440     def note_converter(self, midi_note):
441         """
442         This function convert single midi note value to readable
            ↪  musical note.
443         Important parameter:
444         midi_note: midi note value;
445                    60 ----> C4
446         """
447         if midi_note == '<REST>':
448             return '<REST>'
449         else:
450             if 0 <= midi_note <= 127:
451                 pitches = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G',
                    ↪  '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
                    ↪  127."
460
461     def expand_note_durations(self, s, divisions):
```

```
        for measure in s.getElementsByClass(stream.Measure):
            for element in measure:
                if isinstance(element, music21.note.Note):
                    duration = element.duration.quarterLength
                    expanded_duration = (1 / divisions) * duration
                    offset = element.offset
                    measure.remove(element)
                    for i in range(divisions):
                        new_note = music21.note.Note(element.pitch)
                        new_note.duration.quarterLength =
                        ↪  expanded_duration
                        measure.insert(offset + i * expanded_duration,
                        ↪  new_note)
                elif isinstance(element, music21.chord.Chord):
                    duration = element.duration.quarterLength
                    expanded_duration = (1 / divisions) * duration
                    offset = element.offset
                    measure.remove(element)
                    for i in range(divisions):
                        new_chord =
                        ↪  music21.chord.Chord(element.pitches)
                        for note_in_chord in new_chord:
                            note_in_chord.duration.quarterLength =
                            ↪  expanded_duration
                        measure.insert(offset + i * expanded_duration,
                        ↪  new_chord)
                elif isinstance(element, music21.note.Rest):
                    duration = element.duration.quarterLength
                    expanded_duration = (1 / divisions) * duration
                    offset = element.offset
                    measure.remove(element)
                    for i in range(divisions):
                        new_rest = music21.note.Rest()
                        new_rest.duration.quarterLength =
                        ↪  expanded_duration
                        measure.insert(offset + i * expanded_duration,
                        ↪  new_rest)
```

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

```python
import streamlit as st
import os
from melody_api import Melody
from dynamic_system.dynamic_system import *
import streamlit.components.v1 as components


def main():
    def expanded_index_function(n):
        if n == 2:
            index = 0
        elif n == 4:
            index = 1
        elif n == 8:
            index = 2
        elif n == 16:
            index = 3
        return index

    if 'x' not in st.session_state:
        st.session_state['x'] = 1.01
    if 'y' not in st.session_state:
        st.session_state['y'] = 1.01
    if 'z' not in st.session_state:
        st.session_state['z'] = 1.01
    if 'sigma' not in st.session_state:
        st.session_state['sigma'] = 10.0
    if 'rho' not in st.session_state:
        st.session_state['rho'] = 28.0
    if 'beta' not in st.session_state:
        st.session_state['beta'] = 2.67
    if 'method' not in st.session_state:
        st.session_state['method'] = "Classic"
    if 'option' not in st.session_state:
        st.session_state['option'] = 2
    if 'sequence_select' not in st.session_state:
        st.session_state['sequence_select'] = 0
    if 'uploaded_file' not in st.session_state:
        st.session_state['uploaded_file'] = None
    if 'save_path' not in st.session_state:
        st.session_state['save_path'] = None
    if 'method_index' not in st.session_state:
        st.session_state['method_index'] = 0
    if 'expanded_index' not in st.session_state:
        st.session_state['expanded_index'] = 0
    if 'html_component' not in st.session_state:
        st.session_state['html_component'] = None
    if 'original_path' not in st.session_state:
        st.session_state['original_path'] = None
```

```
    if 'new_path' not in st.session_state:
        st.session_state['new_path'] = None
    if 'midi_new_download_path' not in st.session_state:
        st.session_state['midi_new_download_path'] = None
    if 'pdf_original_path' not in st.session_state:
        st.session_state['pdf_original_path'] = None
    if 'pdf_new_path' not in st.session_state:
        st.session_state['pdf_new_path'] = None
    if 'generate_clicked' not in st.session_state:
        st.session_state['generate_clicked'] = False
    if 'complete_download' not in st.session_state:
        st.session_state['complete_download'] = False
    if 'original_pdf_bytes' not in st.session_state:
        st.session_state['original_pdf_bytes'] = None
    if 'new_midi_bytes' not in st.session_state:
        st.session_state['new_midi_bytes'] = None
    if 'new_pdf_bytes' not in st.session_state:
        st.session_state['new_pdf_bytes'] = None
    if 'track_list' not in st.session_state:
        st.session_state['track_list'] = None
    if 'backup_track_list' not in st.session_state:
        st.session_state['backup_track_list'] = None
    if 'under_process' not in st.session_state:
        st.session_state['under_process'] = None
    if 'file_exist' not in st.session_state:
        st.session_state['file_exist'] = 0

    st.set_page_config(layout="wide")

    # Initialize Melody object
    my_melody = Melody()

    # Page layout
    left_col, right_col = st.columns([1, 3])

    # MIDI file count variable
    file_count = 0

    # Left Column: Generate Music Button and File Upload
    with left_col:

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

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

        css = '''
        <style>
```

```
            [data-testid='stFileUploader'] {
                width: max-content;
            }
            [data-testid='stFileUploader'] section {
                padding: 0;
                float: left;
            }
            [data-testid='stFileUploader'] section > input + div {
                display: none;
            }
            [data-testid='stFileUploader'] section + div {
                display: none;
            }
        </style>
        '''

        st.markdown(css, unsafe_allow_html=True)

        # Only update session state if a new file is uploaded

        if uploaded_file is not None:
            if st.session_state['file_exist'] == 1:
                print(f"now_file = {uploaded_file.name}")
                print(f"exist_file =
                ↪  {st.session_state['uploaded_file'].name}")
                if (uploaded_file.name !=
                ↪  st.session_state['uploaded_file'].name):
                    st.session_state['track_list'] = None
                    st.session_state['backup_track_list'] = None
            st.session_state['uploaded_file'] = uploaded_file
            st.session_state['file_exist'] = 1

        if 'uploaded_file' in st.session_state and
        ↪  st.session_state['uploaded_file'] is not None:
            st.write("Uploaded file:",
            ↪  st.session_state['uploaded_file'].name)

            save_directory = os.path.join("static", "midi_file")
            os.makedirs(save_directory, exist_ok=True)

            if ' ' in str(st.session_state['uploaded_file'].name):
                st.warning('Warning: We can\'t save your MIDI file
                ↪  because your MIDI file name has spacing. Please
                ↪  fill the spacing with "_".')
            else:
                save_path = os.path.join(save_directory,
                ↪  st.session_state['uploaded_file'].name)

                # Reset file pointer to the beginning
                st.session_state['uploaded_file'].seek(0)
```

```
                # Save the uploaded file locally
                with open(save_path, "wb") as f:
                    f.write(st.session_state['uploaded_file'].read())
                st.success("File saved successfully!")

                # Reset file pointer again to read for MIDI processing
                st.session_state['uploaded_file'].seek(0)

                try:
                    # Load MIDI data
                    # midi_data = mido.MidiFile(file=BytesIO(st.sessio
                    ↪  n_state['uploaded_file'].read()))
                    # st.title("MIDI File Information")
                    # st.write(f"Number of Tracks:
                    ↪  {len(midi_data.tracks)}")

                    # Load into your Melody object
                    st.session_state['save_path'] = save_path
                    my_melody.load(path=st.session_state['save_path'])
                except Exception as e:
                    st.error(f"Failed to read MIDI file: {e}")
        generate_botton = st.button('Variate')

    # Right Column: Tabs for Results and Settings
    with right_col:
        tabs = st.tabs(["Results", "Settings"])

        # Results Tab
        with tabs[0]:
            if generate_botton:
                st.session_state['generate_clicked'] = True

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

            if 'uploaded_file' in st.session_state and
            ↪  st.session_state['uploaded_file'] is not None:
                if generate_botton == True:
                    #st.snow()
                    st.session_state['complete_download'] = False

                    #st.write("Waiting for our algorithm to generate a
                    ↪  new variation!")
                    my_melody.set_dynamic_sequence(sequence=st.session
                    ↪  _state['sequence_select'])
                    key_list = my_melody.fit(original_initial_conditio
                    ↪  n=[st.session_state['x'],
                    ↪  st.session_state['y'], st.session_state['z']],
```

```
                    method=st.session_state['method'],
                     ↪  divisions=st.session_state['option'],
                    dynamic=le, d=st.session_state['sigma'],
                     ↪  r=st.session_state['rho'],
                     ↪  b=st.session_state['beta'])
                if st.session_state['track_list'] == None:
                    st.session_state['track_list'] = [key for key
                     ↪  in key_list]
                    st.session_state['backup_track_list'] = [key
                     ↪  for key in key_list]
                my_melody.variate(new_initial_condition=[1.5, 1.5,
                 ↪  1.5], track=st.session_state['track_list'])

                original_path, new_path, midi_new_download_path =
                 ↪  my_melody.export("midi")

                st.session_state['original_path'] = original_path
                st.session_state['new_path'] = new_path

                st.session_state['midi_new_download_path'] =
                 ↪  midi_new_download_path
                with open(st.session_state['midi_new_download_path
                 ↪  '], "rb") as file:
                    st.session_state['new_midi_bytes'] =
                     ↪  file.read()

                pdf_original_path, pdf_new_path =
                 ↪  my_melody.export("pdf")

                st.session_state['pdf_original_path'] =
                 ↪  pdf_original_path
                with open(st.session_state['pdf_original_path'],
                 ↪  "rb") as file:
                    st.session_state['original_pdf_bytes'] =
                     ↪  file.read()

                st.session_state['pdf_new_path'] = pdf_new_path
                with open(st.session_state['pdf_new_path'], "rb")
                 ↪  as file:
                    st.session_state['new_pdf_bytes'] = file.read()

                st.session_state['html_component'] = True

            # MIDI Player and Visualizer
            if 'html_component' in st.session_state and
             ↪  st.session_state['html_component'] is not None:
                components.html(
                f"""
                <h1 style="font-size:30px;">Original Music</h1>
                <center>
                <midi-player
```

```
  src="{st.session_state['original_path']}"
  sound-font visualizer="#myStaffVisualizer1">
</midi-player>

<midi-visualizer type="staff" id="myStaffVisualizer1"
  src="{st.session_state['original_path']}">
</midi-visualizer>
</center>
<script
 ↪  src="https://cdn.jsdelivr.net/combine/npm/tone@14.
 ↪  7.58,npm/@magenta/music@1.23.1/es6/core.js,npm/foc
 ↪  us-visible@5,npm/html-midi-player@1.5.0"></script>
""",
height=300)

original_col1, original_col2, original_col3,
 ↪  original_col4, original_col5 =
 ↪  st.columns([1,1,2,1,1])
with original_col3:
    st.download_button(
        label="Download Original Music Sheet",
        data=st.session_state['original_pdf_bytes'],
        file_name=str(st.session_state['uploaded_file'
         ↪  ].name) + "_original.pdf",
        mime="application/pdf"
        )

components.html(
f"""
<h1 style="font-size:30px;">New Variation Music</h1>
<center>
<midi-player
  src="{st.session_state['new_path']}"
  sound-font visualizer="#myStaffVisualizer2">
</midi-player>

<midi-visualizer type="staff" id="myStaffVisualizer2"
  src="{st.session_state['new_path']}">
</midi-visualizer>
</center>
<script
 ↪  src="https://cdn.jsdelivr.net/combine/npm/tone@14.
 ↪  7.58,npm/@magenta/music@1.23.1/es6/core.js,npm/foc
 ↪  us-visible@5,npm/html-midi-player@1.5.0"></script>
""",
height=300
)

new_col1, new_col2 = st.columns(2)
with new_col1:
    st.download_button(
```

```
                        label="Download MIDI File",
                        data=st.session_state['new_midi_bytes'],
                        file_name=str(st.session_state['uploaded_file'
                        ↪  ].name) + "_new.mid",
                        mime="audio/midi"
                    )
                with new_col2:
                    st.download_button(
                        label="Download New Music Sheet",
                        data=st.session_state['new_pdf_bytes'],
                        file_name=str(st.session_state['uploaded_file'
                        ↪  ].name) + "_new.pdf",
                        mime="application/pdf"
                    )
                st.session_state['complete_download'] = True
                st.write("Complete!")


        # Settings Tab
        with tabs[1]:
            st.title("Settings")

            equation = st.selectbox(
                "Choose Chaotic Systems",
                ("Lorenz system", "Chaotic system"),
                index=0,
                placeholder="Choose Equation"
            )

            # Actions for Lorenz system
            if equation == "Lorenz system":
                # Variable selection
                variable_Lorenz = st.selectbox(
                    "Select variable to display",
                    options=("x", "y", "z"),
                    index=0
                )

                # Create a two-column layout
                col1, col2 = st.columns(2)

                with col1:
                    # Display Variables
                    st.markdown("### Variables")

                    # Sliders for variables
                    st.session_state['x'] = st.slider(
                        "x",
                        min_value=-5.0,
                        max_value=5.0,
                        value=1.01,
```

```
                step=0.01
            )
            st.session_state['y'] = st.slider(
                "y",
                min_value=-5.0,
                max_value=5.0,
                value=1.01,
                step=0.01
            )
            st.session_state['z'] = st.slider(
                "z",
                min_value=-5.0,
                max_value=5.0,
                value=1.01,
                step=0.01
            )

            # Set the variable based on selection
            if variable_Lorenz == "x":
                st.session_state['sequence_select'] = 0
            elif variable_Lorenz == "y":
                st.session_state['sequence_select'] = 1
            elif variable_Lorenz == "z":
                st.session_state['sequence_select'] = 2

        with col2:
            # Display Parameters
            st.markdown("### Parameters")

            # Sliders for parameters
            st.session_state['sigma'] = st.slider(
                r"$\sigma$",
                min_value=0.0,
                max_value=100.0,
                value=10.0,
                step=0.01
            )
            st.session_state['rho'] = st.slider(
                r"$\rho$",
                min_value=0.0,
                max_value=100.0,
                value=28.0,
                step=0.01
            )
            st.session_state['beta'] = st.slider(
                r"$\beta$",
                min_value=0.0,
                max_value=100.0,
                value=2.67,
                step=0.01
            )
```

```
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
                        ↪ _state['sequence_select'])
368                     my_melody.fit(initial_condition=[st.session_state[
                        ↪ 'x'],st.session_state['y'],st.session_state['z
                        ↪ ']], dynamic=le, d=st.session_state['sigma'],
                        ↪ r=st.session_state['rho'],
                        ↪ b=st.session_state['beta'])
369                     checker = my_melody.is_chaotic()
370                     if str(checker) == 'True':
371                         st.write(f"Chaotic parameter")
372                     else:
373                         st.warning('Parameter not chaotic')
374
375             elif equation == "Chaotic system":
376                 st.write("Soon...")
377             if (st.session_state['complete_download'] == True) and
                ↪ (st.session_state['backup_track_list'] != None):
378                 options = st.multiselect(
379                     "Select MIDI Tracks",
380                     list(st.session_state['backup_track_list']),
381                     default =
                        ↪ list(st.session_state['backup_track_list'])
382                 )
383                 st.write(list(options))
384                 #print(list(options))
385                 st.session_state['track_list'] = list(options)
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
                        ↪ algorithm will expand 1 note into 4 notes in
                        ↪ your MIDI file)",
403                     (2, 4, 8, 16),
404                     index=st.session_state['expanded_index'],
```

```
                )
                st.session_state['expanded_index'] =
                ↪  expanded_index_function(st.session_state['option'])

if __name__ == "__main__":
    main()
```

melody_api.py

```
import music21
from music21 import stream, midi, scale, converter, instrument
from datetime import datetime as dt
import numpy as np
import subprocess
import os
import pathlib

class Melody:

    def __init__(self):
        self.tracks = {}
        self.maps = {}
        self.dynamic = None

        # Main Instance Attributes
        self.original_midi_data = None
        self.new_midi_data = None

        # Additional Instance Attributes
        self.newtracks = {}
        self.main_stream = None
        self.new_stream = None
        self.duration = {}
        self.new_duration = {}
        self.initial_condition = None
        self.dynamic_sequence = 0
        self.notetrack = {}
        self.tracks_index = {}
        self.dynamic_parameter = {}

        # Support Instance Attributes
        self.method = None
        self.divisions = None
        self.backtrack_expand_index = []

        # Path
        self.lilypond_path =
        ↪  str(pathlib.Path("__file__").parent.resolve()) +
        ↪  "\\lilypond-2.24.4\\bin\\lilypond.exe"

        # etc.
```

```
        self.filename = None
        self.dummy1 = None
        self.dummy2 = None
        self.dummy3 = None
        self.dummy4 = None

    ########################
    # Main Method
    ########################

    def load(self, path):
        midi_file = midi.MidiFile()
        midi_file.open(path)
        midi_file.read()
        midi_file.close

        self.main_stream = midi.translate.midiFileToStream(midi_file)
        self.new_stream = midi.translate.midiFileToStream(midi_file)

        self.original_midi_data =
        ↪  midi.translate.midiFileToStream(midi_file)
        self.new_midi_data = midi.translate.midiFileToStream(midi_file)

        self.dummy1 = midi.translate.midiFileToStream(midi_file)
        self.dummy2 = midi.translate.midiFileToStream(midi_file)
        self.dummy3 = midi.translate.midiFileToStream(midi_file)
        self.dummy4 = midi.translate.midiFileToStream(midi_file)

        self.filename = str(pathlib.Path(path).stem)
        #return self.tracks

    def fit(self, original_initial_condition, dynamic,
    ↪  method="Classic", divisions=None, *args, **kwargs):
        self.original_initial_condition = original_initial_condition
        self.dynamic_parameter.update(*args, **kwargs)
        self.dynamic = dynamic
        self.method = method
        self.divisions = divisions

        if self.method == "Classic":
            for index in range(len(self.main_stream)):
                instruments = self.main_stream[index][0].getElementsBy」
                ↪  Class(instrument.Instrument)
                if instruments:
                    instrument_name =
                    ↪  str(instruments[0].instrumentName) + "_" +
                    ↪  str(index)
                else:
                    instrument_name = "track_" + str(index)
                music_list = []
                duration_list = []
```

```
                for part in self.main_stream[index]:
                    for n in part.flat.notesAndRests:
                        dur = n.duration.quarterLength
                        if n.isRest:
                            note_num = "<REST>"
                            music_list.append(note_num)
                            duration_list.append(dur)
                        else:
                            if len(n.pitches) == 1:
                                note_num = n.pitches[0].midi
                                music_list.append(note_num)
                                duration_list.append(dur)
                            else:
                                note_num = sorted(set([p.midi for p in
                                 ↪  n.pitches]))
                                music_list.append(list(note_num))
                                duration_list.append(dur)
                string_music_list = [element for element in music_list]
                for i in range(len(music_list)):
                    try:
                        string_music_list[i] =
                         ↪  self.note_converter(music_list[i])
                    except TypeError:
                        for j in range(len(music_list[i])):
                            string_music_list[i][j] =
                             ↪  self.note_converter(music_list[i][j])

                time, solution = self.rk4(time=[0,
                 ↪  0.01*(len(music_list) - 1)], f=self.dynamic,
                 ↪  ini=self.original_initial_condition,
                 ↪  **self.dynamic_parameter)
                main_sequence = solution[:, self.dynamic_sequence]
                #self.original_time = time
                # self.original_trajectory = main_sequence

                self.notetrack.update({instrument_name:string_music_li
                 ↪  st})
                self.duration.update({instrument_name:duration_list})
                self.tracks.update({instrument_name:main_sequence})
                self.tracks_index.update({instrument_name:index})
            for key in self.tracks.keys():
                dummy_list = []
                for i in range(len(self.notetrack[key])):
                    dummy_list.append([self.tracks[key][i],
                     ↪  self.notetrack[key][i]])
                self.maps.update({key:dummy_list})
        elif self.method == "Expanded":
            for index in range(len(self.main_stream)):
                instruments = self.main_stream[index][0].getElementsBy
                 ↪  Class(instrument.Instrument)
                if instruments:
```

```
                    instrument_name =
                    ↪ str(instruments[0].instrumentName) + "_" +
                    ↪ str(index)
                else:
                    instrument_name = "track_" + str(index)
                duration_list = []
                for part in self.main_stream[index]:
                    for n in part.flat.notesAndRests:
                        dur = n.duration.quarterLength
                        if n.isRest:
                            duration_list.append(dur)
                        else:
                            if len(n.pitches) == 1:
                                duration_list.append(dur)
                            else:
                                duration_list.append(dur)
                self.duration.update({instrument_name:duration_list})
            for index in range(len(self.main_stream)):
                self.expand_note_durations(s=self.main_stream[index],
                ↪ divisions=self.divisions)
                instruments = self.main_stream[index][0].getElementsBy」
                ↪ Class(instrument.Instrument)
                if instruments:
                    instrument_name =
                    ↪ str(instruments[0].instrumentName) + "_" +
                    ↪ str(index)
                else:
                    instrument_name = "track_" + str(index)
                music_list = []
                #duration_list = []
                for part in self.main_stream[index]:
                    for n in part.flat.notesAndRests:
                        dur = n.duration.quarterLength
                        if n.isRest:
                            note_num = "<REST>"
                            music_list.append(note_num)
                            #duration_list.append(dur)
                        else:
                            if len(n.pitches) == 1:
                                note_num = n.pitches[0].midi
                                music_list.append(note_num)
                                #duration_list.append(dur)
                            else:
                                note_num = sorted(set([p.midi for p in
                                ↪ n.pitches]))
                                music_list.append(list(note_num))
                                #duration_list.append(dur)
                string_music_list = [element for element in music_list]
                for i in range(len(music_list)):
                    try:
```

```
                        string_music_list[i] =
                        ↪ self.note_converter(music_list[i])
                    except TypeError:
                        for j in range(len(music_list[i])):
                            string_music_list[i][j] =
                            ↪ self.note_converter(music_list[i][j])
                self.backtrack_expand_index = [index for index in
                ↪ range(0, len(music_list) - 1, self.divisions)]
                time, solution = self.rk4(time=[0,
                ↪ 0.01*(len(music_list) - 1)], f=self.dynamic,
                ↪ ini=self.original_initial_condition,
                ↪ **self.dynamic_parameter)
                main_sequence = solution[:, self.dynamic_sequence]
                #self.original_time = time
                # self.original_trajectory = main_sequence

                self.notetrack.update({instrument_name:string_music_li」
                ↪ st})
                self.tracks.update({instrument_name:main_sequence})
                self.tracks_index.update({instrument_name:index})
            for key in self.tracks.keys():
                dummy_list = []
                for i in range(len(self.notetrack[key])):
                    dummy_list.append([self.tracks[key][i],
                    ↪ self.notetrack[key][i]])
                self.maps.update({key:dummy_list})
        else:
            raise Exception("Out of Option!")

        #return self.maps
        return self.tracks.keys()

    def variate(self, new_initial_condition, track=None,
    ↪ criteria="right", add_note=0):
        self.new_initial_condition = new_initial_condition
        if track == None:
            track = list(self.tracks.keys())
        if add_note != 0:
            for key in track:
                for each_note in range(add_note):
                    # Access parts from the stream
                    parts = self.new_stream.getElementsByClass('Part')
                    target_part = parts[self.tracks_index[key]]

                    # Determine the last measure in the part
                    if target_part.measure(-1):
                        last_measure = target_part.measure(-1)  # Get
                        ↪ the last measure
                    else:
                        # If no measures exist, create the first one
                        last_measure = music21.stream.Measure()
```

```
                        target_part.append(last_measure)

                    # Create the new note
                    new_note = music21.note.Note('C4')
                    duration_len = len(self.duration[key])
                    new_note.quarterLength =
                     ↪  self.duration[key][each_note % duration_len]

                    # Append the new note to the last measure
                    last_measure.append(new_note)
                #print(target_part[-1].show("text"))
                #print("-"*100)

        if self.method == "Expanded":
            for index in range(len(self.new_stream)):
                self.expand_note_durations(s=self.new_stream[index],
                 ↪  divisions=self.divisions)
                instruments = self.new_stream[index][0].getElementsByC
                 ↪  lass(instrument.Instrument)
                if instruments:
                    instrument_name =
                     ↪  str(instruments[0].instrumentName) + "_" +
                     ↪  str(index)
                else:
                    instrument_name = "track_" + str(index)
                duration_list = []
                for part in self.new_stream[index]:
                    for n in part.flat.notesAndRests:
                        dur = n.duration.quarterLength
                        duration_list.append(dur)
                self.new_duration.update({instrument_name:duration_lis
                 ↪  t})
            #self.duration = self.new_duration

        for key in track:
            sorted_tracks = sorted(self.maps[key], key=lambda x: x[0])
            if self.method == "Classic":
                dummy_list = [None]*(len(self.tracks[key]) + add_note)
                time, solution = self.rk4(time=[0,
                 ↪  0.01*(len(dummy_list) - 1)], f=self.dynamic,
                 ↪  ini=self.new_initial_condition,
                 ↪  **self.dynamic_parameter)
                main_sequence = solution[:, self.dynamic_sequence]
            elif self.method == "Expanded":
                dummy_list = [None]*(len(self.tracks[key]) +
                 ↪  add_note)*self.divisions
                time, solution = self.rk4(time=[0,
                 ↪  0.01*(len(dummy_list) - 1)], f=self.dynamic,
                 ↪  ini=self.new_initial_condition,
                 ↪  **self.dynamic_parameter)
                main_sequence = solution[:, self.dynamic_sequence]
```

```
                # self.new_time = time
                # self.new_trajectory = main_sequence
            else:
                raise Exception("Out of Option!")

            if criteria == "right":
                for i in range(len(main_sequence) if len(main_sequence) == len(dummy_list) else len(main_sequence) - 1):
                    for j in range(len(sorted_tracks)):
                        if main_sequence[i] <= sorted_tracks[j][0]:
                            dummy_list[i] = sorted_tracks[j][1]
                            break
                        elif main_sequence[i] > sorted_tracks[-1][0]:
                            dummy_list[i] = sorted_tracks[-1][1]
                            break
            elif criteria == "left":
                for i in range(len(main_sequence) if len(main_sequence) == len(dummy_list) else len(main_sequence) - 1):
                    for j in range(len(sorted_tracks)):
                        if main_sequence[i] <= sorted_tracks[j][0]:
                                dummy_list[i] = sorted_tracks[j - 1][1]
                                break
                        elif main_sequence[i] > sorted_tracks[-1][0]:
                            dummy_list[i] = sorted_tracks[-1][1]
                            break
                        elif (main_sequence[i] <= sorted_tracks[0][0]) and (j - 1 < 0):
                            main_sequence[i] = sorted_tracks[0][0]
                            break
            else:
                raise Exception("Out of Option!")

            self.newtracks.update({key:dummy_list})

    def export(self, format_type):
        exist_track = [track for track in self.newtracks.keys()]
        exist_new_stream = self.new_midi_data
        if self.method == "Classic":
            for key in self.tracks.keys():
                if key in exist_track:
                    index = 0
                    for element in exist_new_stream[self.tracks_index[key]].recurse():
                        if isinstance(element, (music21.note.Note, music21.chord.Chord, music21.note.Rest)):
                            replacement = self.newtracks[key][index]
                            duration_value = self.duration[key][index]
                            if replacement == '<REST>':
                                new_element = music21.note.Rest()
                            elif isinstance(replacement, list):
```

```
                                new_element =
                                    ↪ music21.chord.Chord(replacement)
                            else:
                                new_element =
                                    ↪ music21.note.Note(replacement)
                            new_element.duration = music21.duration.Du
                            ↪ ration(duration_value)
                            element.activeSite.replace(element,
                            ↪ new_element)
                            print(f"{element} --> {replacement}")
                            if index < len(self.newtracks[key]) - 1:
                                index += 1
                            else:
                                break
        elif self.method == "Expanded":
            for key in self.tracks.keys():
                if key in exist_track:
                    index = 0
                    for element in exist_new_stream[self.tracks_index[
                    ↪ key]].recurse():
                        if isinstance(element, (music21.note.Note,
                        ↪ music21.chord.Chord, music21.note.Rest)):
                            replacement = self.newtracks[key][self.bac
                            ↪ ktrack_expand_index[index]]
                            duration_value = self.duration[key][index]
                            if replacement == '<REST>':
                                new_element = music21.note.Rest()
                            elif isinstance(replacement, list):
                                new_element =
                                    ↪ music21.chord.Chord(replacement)
                            else:
                                new_element =
                                    ↪ music21.note.Note(replacement)
                            new_element.duration = music21.duration.Du
                            ↪ ration(duration_value)
                            element.activeSite.replace(element,
                            ↪ new_element)
                            #print(f"{element} --> {replacement}")
                            if index < len(self.backtrack_expand_index)
                            ↪ - 1:
                                index += 1
                            else:
                                break
        else:
            raise Exception("Out of Option!")
        current_path = pathlib.Path("__file__").parent.resolve()
        date = str(dt.now().isoformat())[:-7]
        filepath = str(current_path) + "\\" + "static" + "\\" +
        ↪ "file_store" + "\\" + self.filename + date.replace(":",
        ↪ "-")
        isExist = os.path.exists(filepath)
```

```
        if not isExist:
            os.makedirs(filepath)
        original = self.original_midi_data
        new = self.new_midi_data
        streamlit_accept_path = "app" + "/" + "static" + "/" +
        ↪  "file_store" + "/" + self.filename + date.replace(":", "-")
        if format_type == "midi":
            # original.metadata = music21.metadata.Metadata()
            original.write('midi', filepath + "\\" + self.filename +
            ↪  '_original.mid')
            original_path = streamlit_accept_path + "/" + self.filename
            ↪  + '_original.mid'
            # new.metadata = music21.metadata.Metadata()
            if self.method == "Classic":
                new.write('midi', filepath + "\\" + self.filename +
                ↪  '_new.mid')
                new_path = streamlit_accept_path + "/" + self.filename
                ↪  + '_new.mid'
                midi_new_download_path = filepath + "\\" +
                ↪  self.filename + '_new.mid'
            elif self.method == "Expanded":
                new.write('midi', filepath + "\\" + self.filename +
                ↪  '_new_expanded.mid')
                new_path = streamlit_accept_path + "/" + self.filename
                ↪  + '_new_expanded.mid'
                midi_new_download_path = filepath + "\\" +
                ↪  self.filename + '_new_expanded.mid'
            return original_path , new_path, midi_new_download_path
        elif format_type == "pdf":
            us = music21.environment.UserSettings()
            us['lilypondPath'] = self.lilypond_path

            original_path = filepath + "\\" + self.filename +
            ↪  '_original.pdf'
            original.metadata = music21.metadata.Metadata()
            original.metadata.title = self.filename.replace("_", " ") +
            ↪  " Original"
            conv = music21.converter.subConverters.ConverterLilypond()
            original_filepath = filepath + "\\" + self.filename +
            ↪  '_original'
            conv.write(original, fmt='lilypond', fp=original_filepath,
            ↪  subformats = ['pdf'])

            self.main_stream = self.dummy1
            self.original_midi_data = self.dummy2

            new_path = filepath + "\\" + self.filename + '_new.pdf'
            new.metadata = music21.metadata.Metadata()
            new.metadata.title = self.filename.replace("_", " ") + "
            ↪  New"
```

```
            new_conv =
             ↪  music21.converter.subConverters.ConverterLilypond()
            new_filepath = filepath + "\\" + self.filename + '_new'
            conv.write(new, fmt='lilypond', fp=new_filepath, subformats
             ↪  = ['pdf'])

            self.new_stream = self.dummy3
            self.new_midi_data = self.dummy4
            return original_path , new_path

    ##########################
    # Additional Method
    ##########################

    def set_dynamic_sequence(self, sequence):
        self.dynamic_sequence = sequence

    ##########################
    # Support Method
    ##########################

    def rk4(self, time, ini, f, h=0.01, *args, **kwargs):
        t = np.arange(time[0], time[1]+h, h)
        row = len(t)
        try:
            var = len(ini)
        except TypeError:
            var = 1
        x = np.zeros((row, var))
        x[0, :] = ini
        for j in range(0, len(t)-1):
            k1 = f(t[j], x[j], *args, **kwargs)
            k2 = f(t[j] + 0.5*h, x[j] +0.5*h*k1, *args, **kwargs)
            k3 = f(t[j] + 0.5*h, x[j] + 0.5*h*k2, *args, **kwargs)
            k4 = f(t[j] + h, x[j] + h*k3, *args, **kwargs)
            x[j+1, :] = x[j] + (h/6)*(k1 + 2*k2 + 2*k3 + k4)
        return t, x

    def is_chaotic(self):
        T = [0, 100]

        # Initial state and parameters
        initial_state = np.array([1.0, 1.0, 1.0])

        # Integrate system
        #sol = solve_ivp(lorenz, t_span, initial_state, t_eval=t_eval,
         ↪  rtol=1e-9, atol=1e-9)
        time, sol = self.rk4(time=T, ini=initial_state, f=self.dynamic,
         ↪  **self.dynamic_parameter)

        # Perturb the initial state slightly
```

```
        perturbed_state = np.array([1.00001, 1.0, 1.0])

        #perturbed_sol = solve_ivp(lorenz, t_span, perturbed_state,
        ↪  t_eval=t_eval, rtol=1e-9, atol=1e-9)
        time, perturbed_sol = self.rk4(time=T, ini=perturbed_state,
        ↪  f=self.dynamic, **self.dynamic_parameter)

        # Compute divergence between trajectories
        divergence = np.linalg.norm(sol - perturbed_sol, axis=1)

        # Estimate Lyapunov exponent
        divergence_rate = np.log(divergence + 1e-10)  # Avoid log(0)
        lyapunov_exponent = np.mean(divergence_rate[time > 10])  #
        ↪  Ignore transient phase

        # Chaos detected if Lyapunov exponent is positive
        return lyapunov_exponent > 0

    def note_converter(self, midi_note):
        """
        This function convert single midi note value to readable
        ↪  musical note.
        Important parameter:
        midi_note: midi note value;
                   60 ----> C4
        """
        if midi_note == '<REST>':
            return '<REST>'
        else:
            if 0 <= midi_note <= 127:
                pitches = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G',
                ↪  'G#', 'A', 'A#', 'B']

                octave = (midi_note // 12) - 1
                pitch_index = midi_note % 12

                pitch = pitches[pitch_index]
                return f"{pitch}{octave}"
            else:
                return "Invalid MIDI note value. Must be between 0 and
                ↪  127."

    def expand_note_durations(self, s, divisions):
        for measure in s.getElementsByClass(stream.Measure):
            for element in measure:
                if isinstance(element, music21.note.Note):
                    duration = element.duration.quarterLength
                    expanded_duration = (1 / divisions) * duration
                    offset = element.offset
                    measure.remove(element)
                    for i in range(divisions):
```

```
                            new_note = music21.note.Note(element.pitch)
                            new_note.duration.quarterLength =
                             ↪  expanded_duration
                            measure.insert(offset + i * expanded_duration,
                             ↪  new_note)
                    elif isinstance(element, music21.chord.Chord):
                        duration = element.duration.quarterLength
                        expanded_duration = (1 / divisions) * duration
                        offset = element.offset
                        measure.remove(element)
                        for i in range(divisions):
                            new_chord =
                             ↪  music21.chord.Chord(element.pitches)
                            for note_in_chord in new_chord:
                                note_in_chord.duration.quarterLength =
                                 ↪  expanded_duration
                            measure.insert(offset + i * expanded_duration,
                             ↪  new_chord)
                    elif isinstance(element, music21.note.Rest):
                        duration = element.duration.quarterLength
                        expanded_duration = (1 / divisions) * duration
                        offset = element.offset
                        measure.remove(element)
                        for i in range(divisions):
                            new_rest = music21.note.Rest()
                            new_rest.duration.quarterLength =
                             ↪  expanded_duration
                            measure.insert(offset + i * expanded_duration,
                             ↪  new_rest)
```

dynamic_system.py

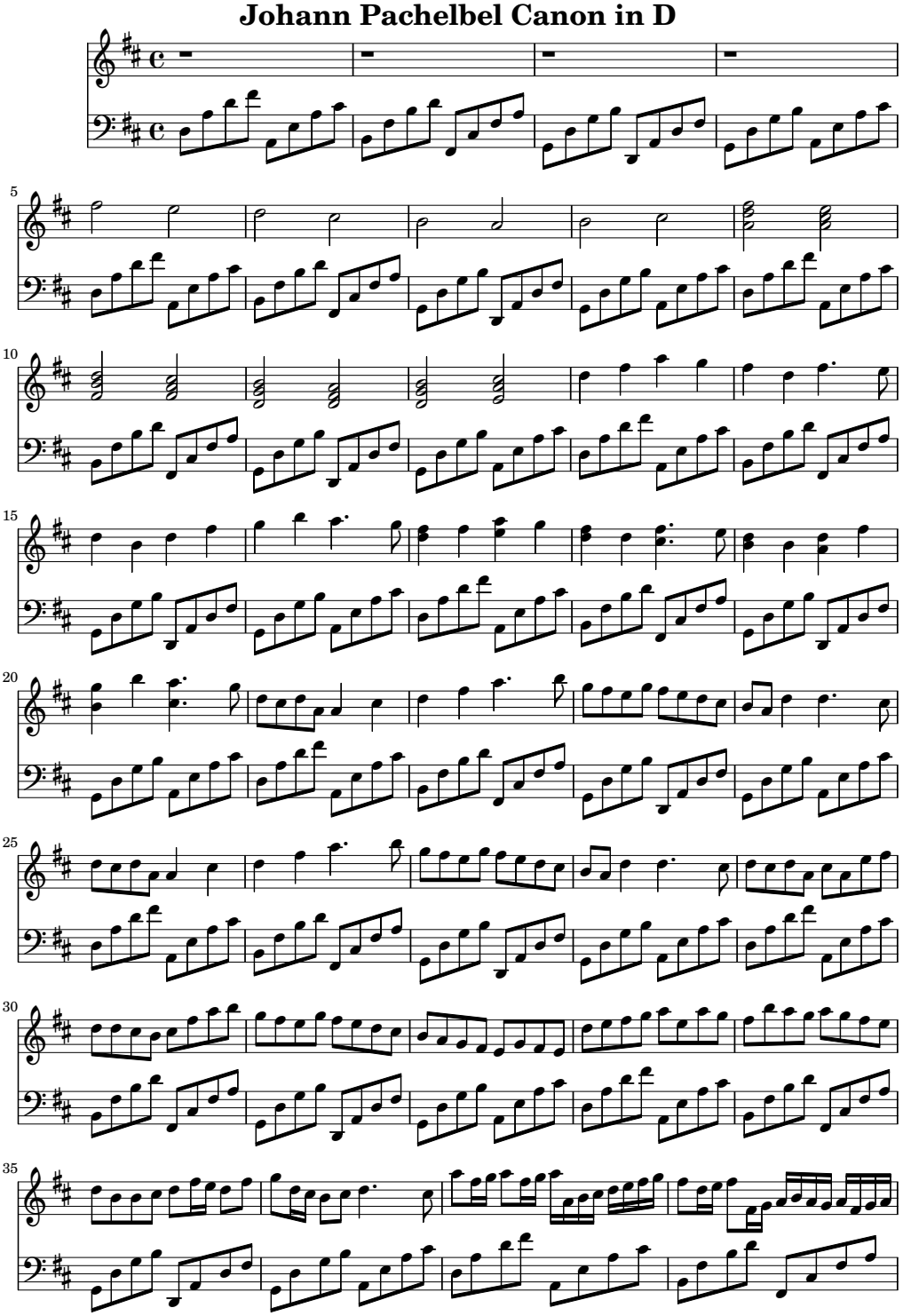
```
import numpy as np

def le(t, xyz, *args, **kwargs):
        """
        ****************************************************************
         ↪  ******************
        "I didn't have any knowledge about Chaotic System" you say,
        Nah Just use this one and change to any chaotic system you want
         ↪  later.
        Chaotic Trajectory parameter:
        d: 10
        r: 28
        b: 8/3
        ****************************************************************
         ↪  ******************
        !!! This function relate with rk4 !!!
        """
        d, r, b = kwargs['d'], kwargs['r'], kwargs['b']
        x = xyz[0]
```

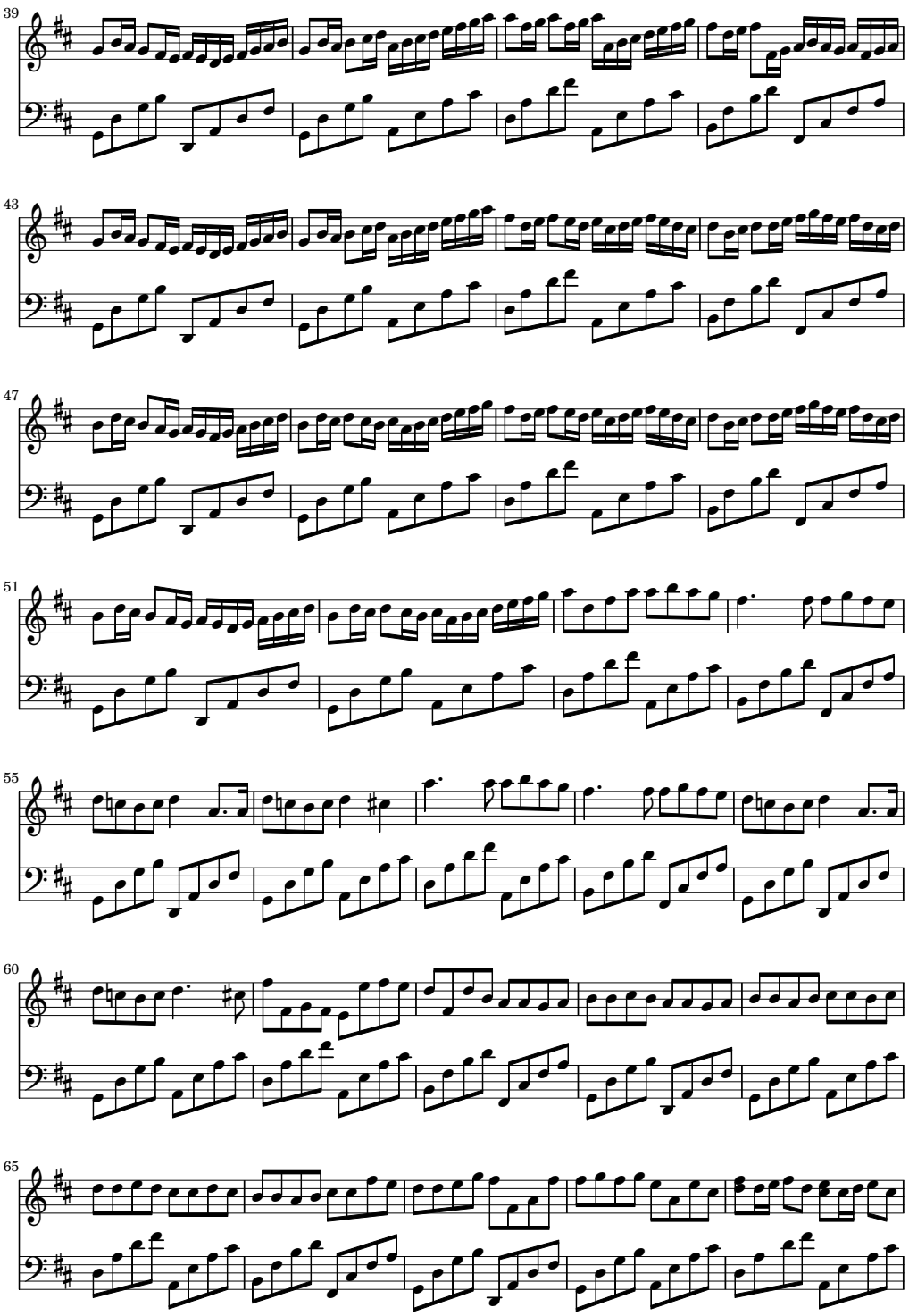
```
17          y = xyz[1]
18          z = xyz[2]
19          xdot = d*(y - x)
20          ydot = x*(r - z) - y
21          zdot = x*y - b*z
22          return np.array([xdot, ydot, zdot])
```

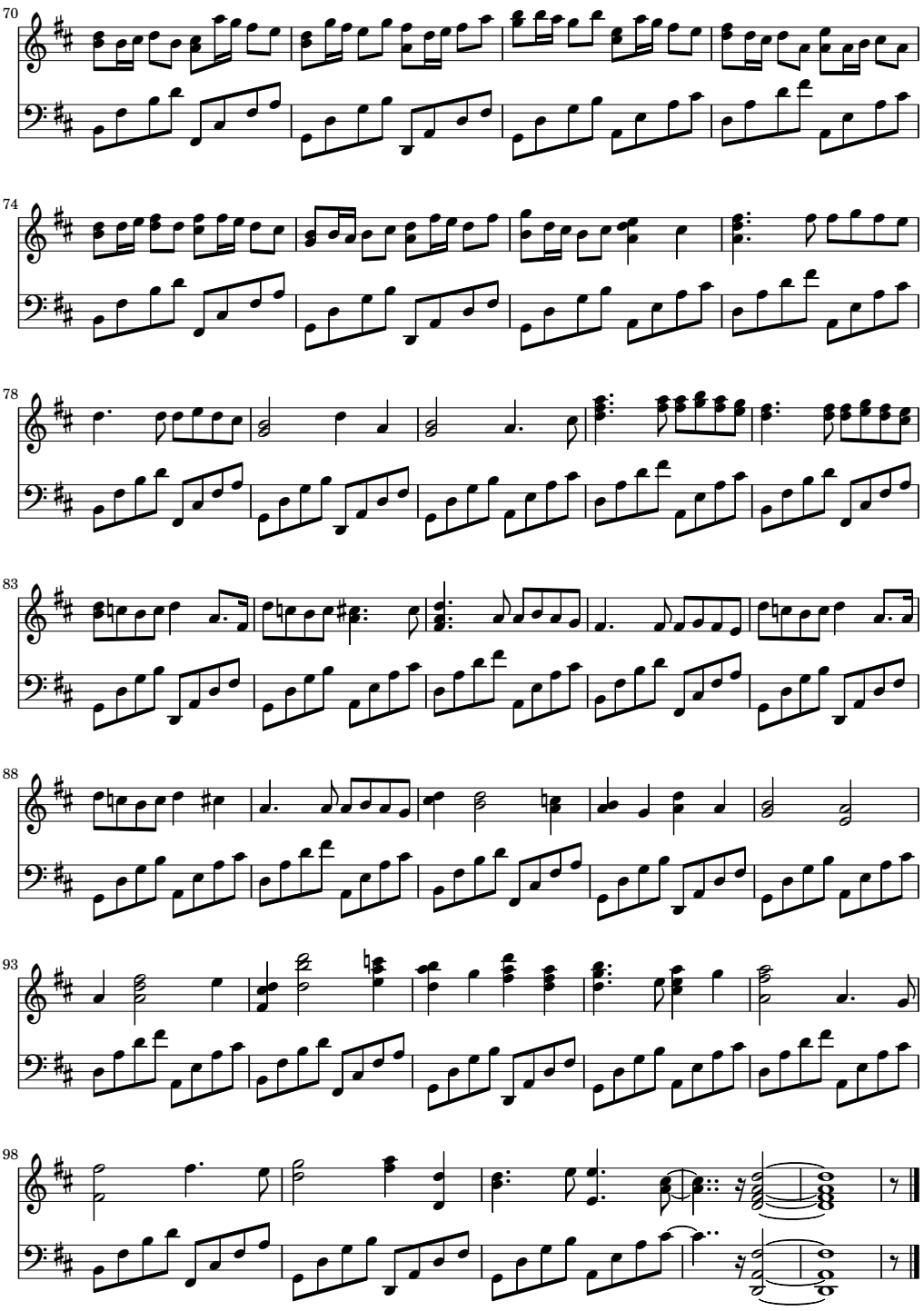
Music Sheets

Pachelbel's Canon

Original Music Sheet

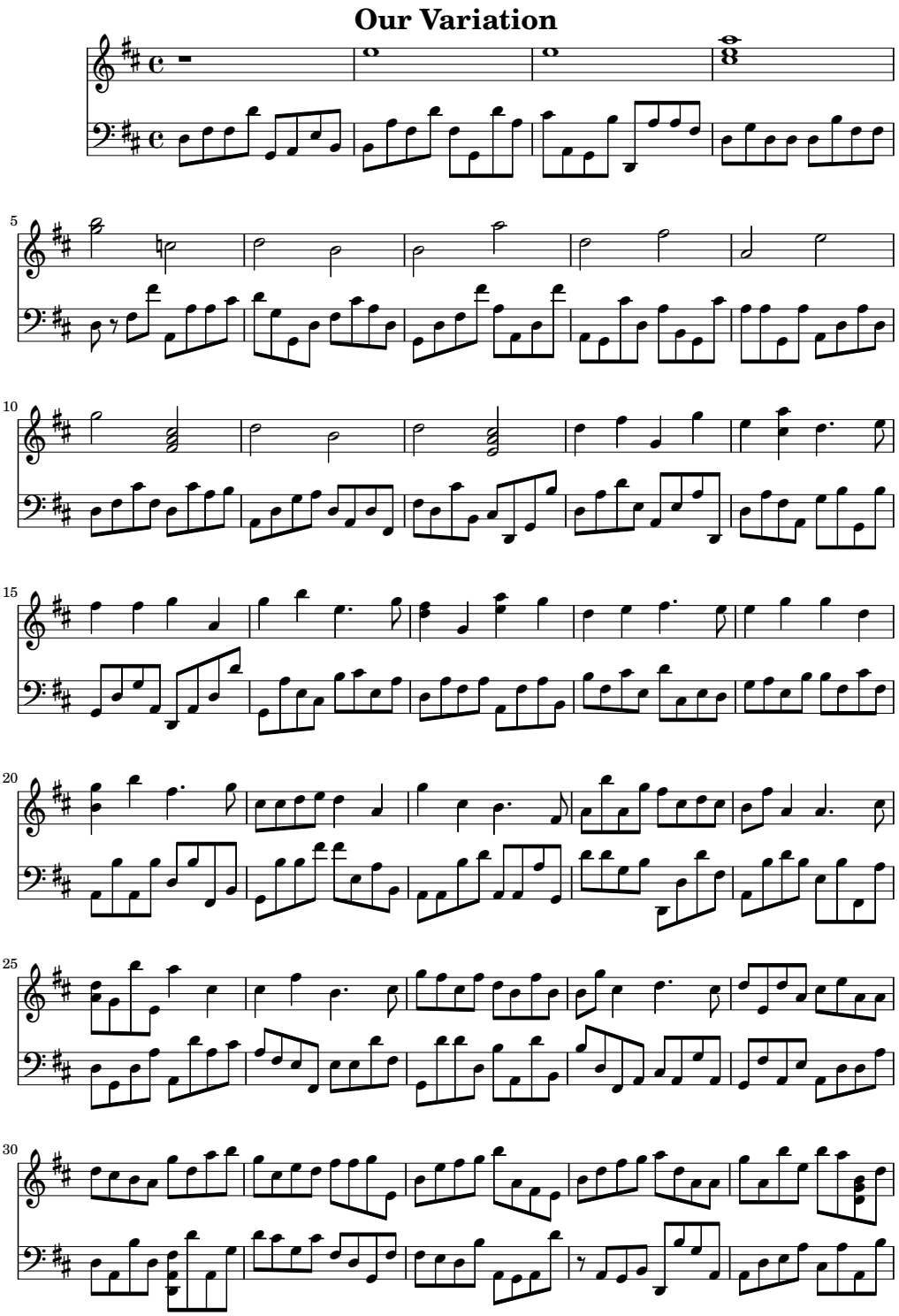


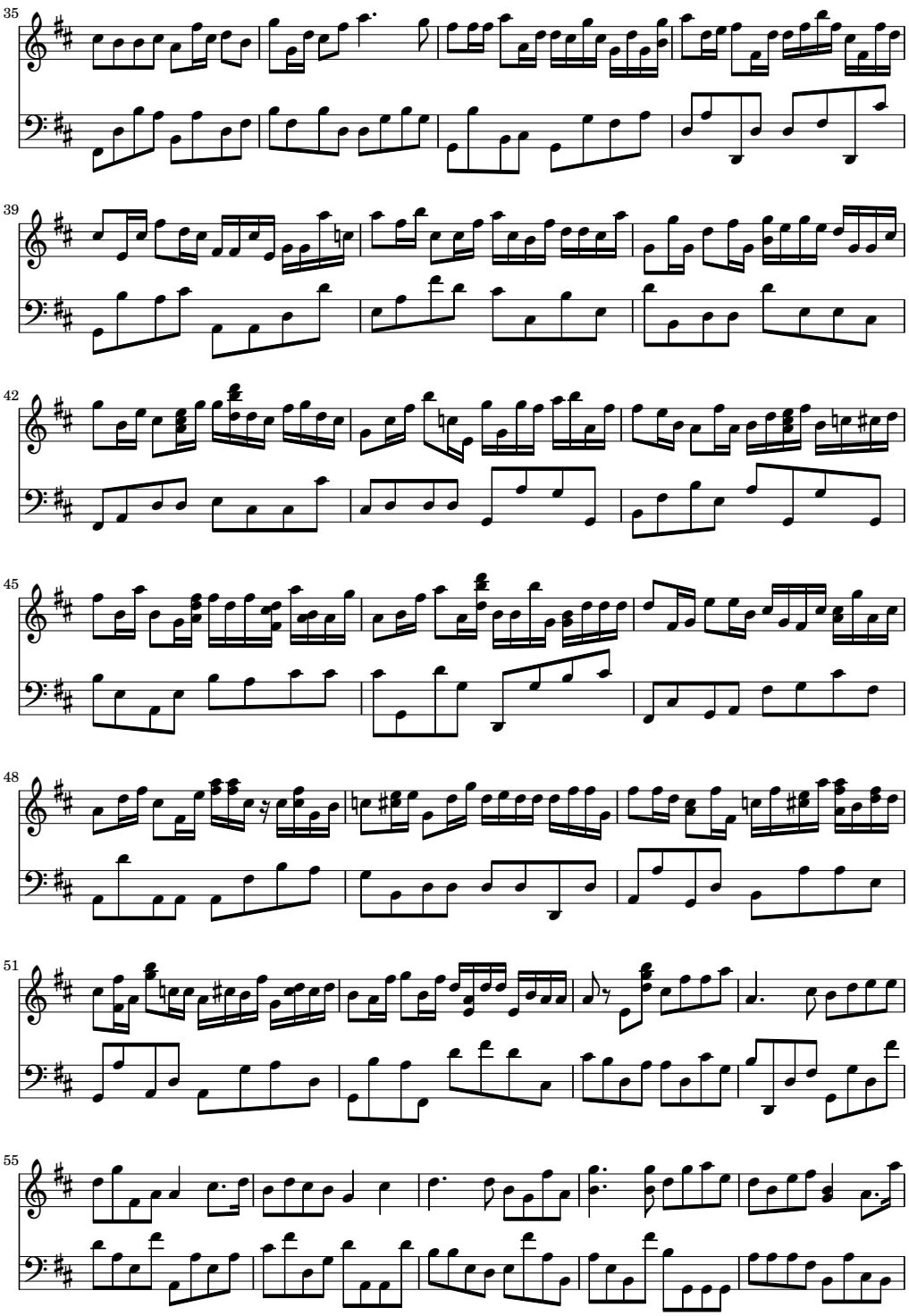

2
39
43
47
51
55
60
65

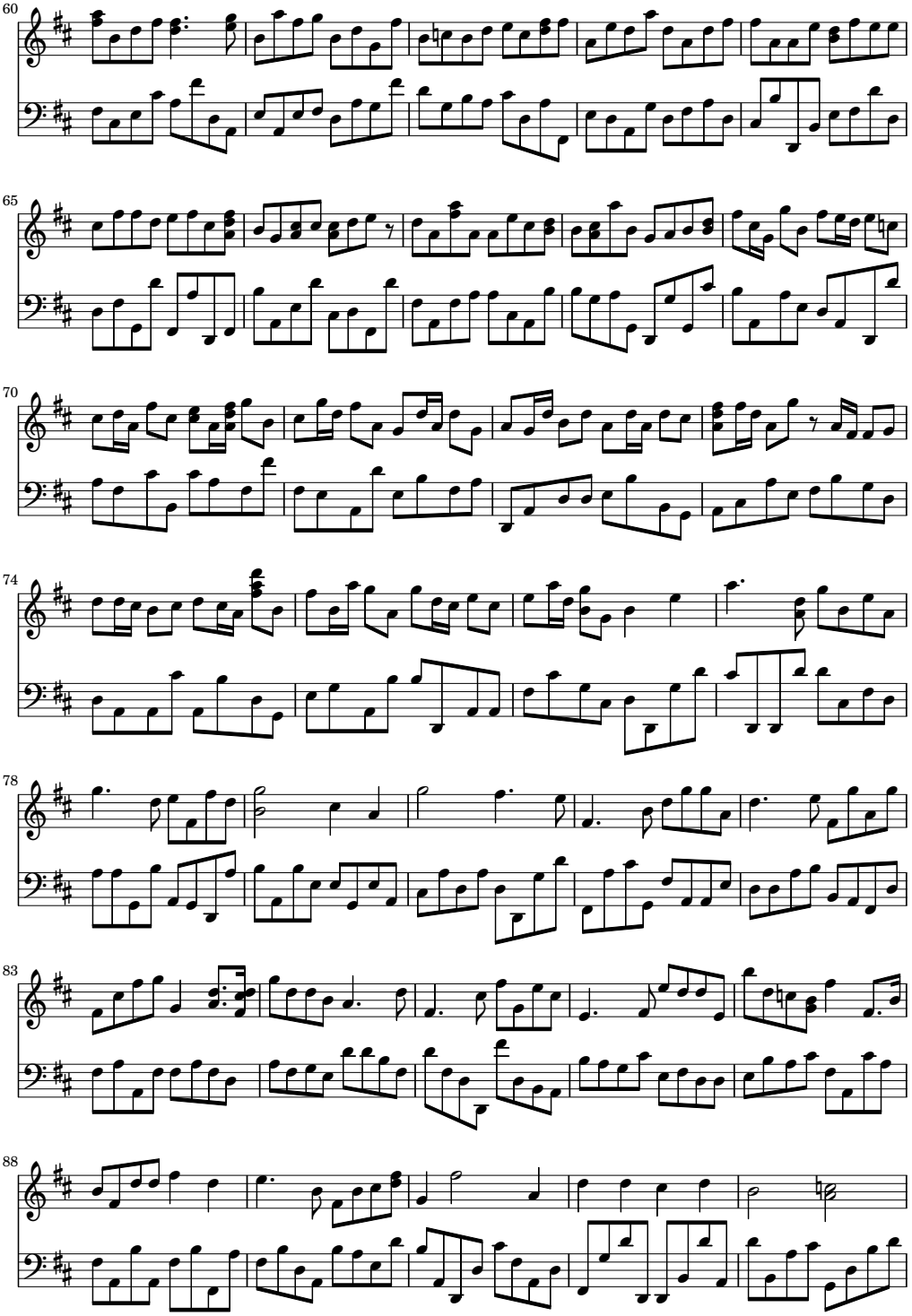


Music engraving by LilyPond 2.24.4—www.lilypond.org

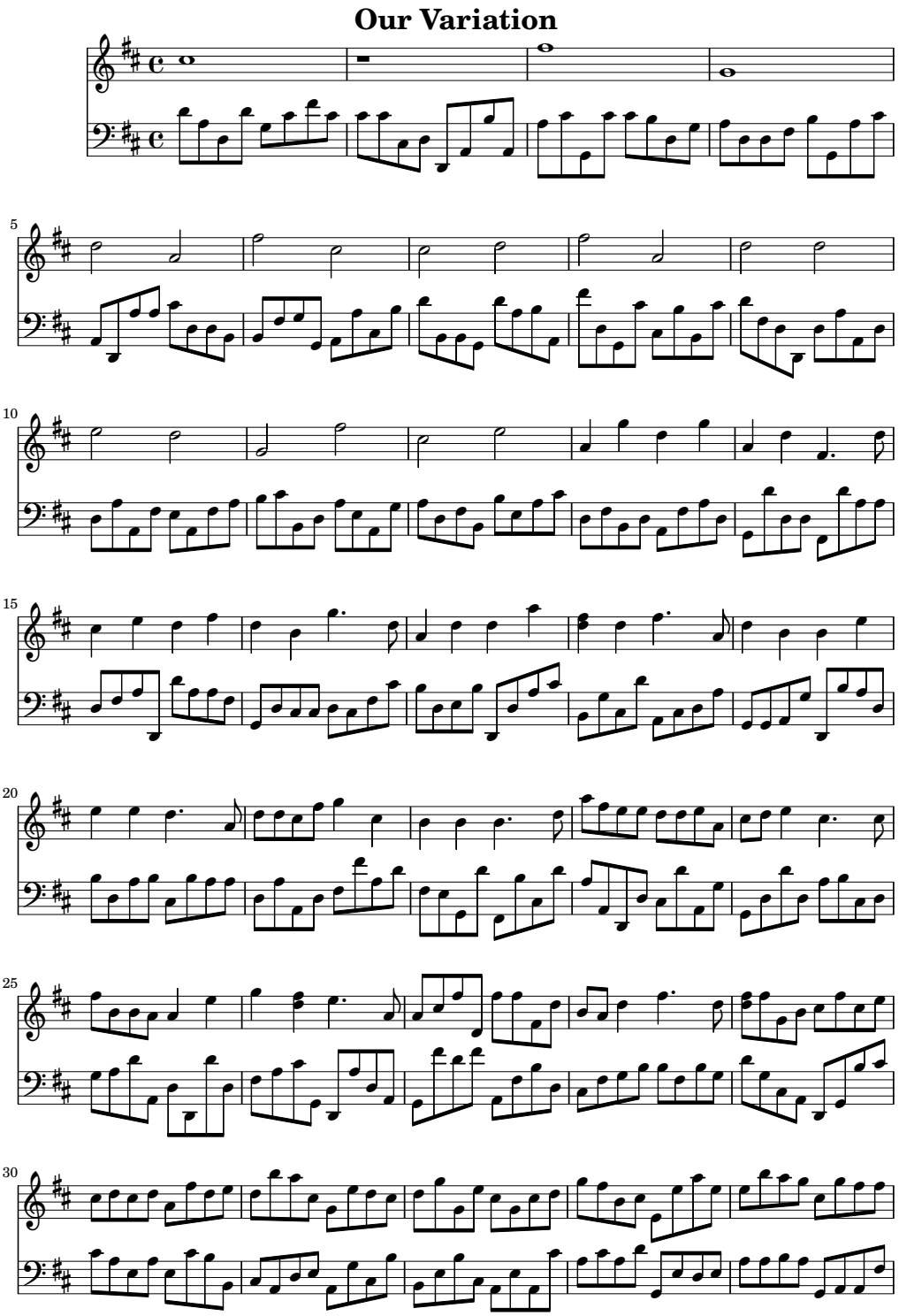
New Variaition from Mapping $\vec{\beta}$

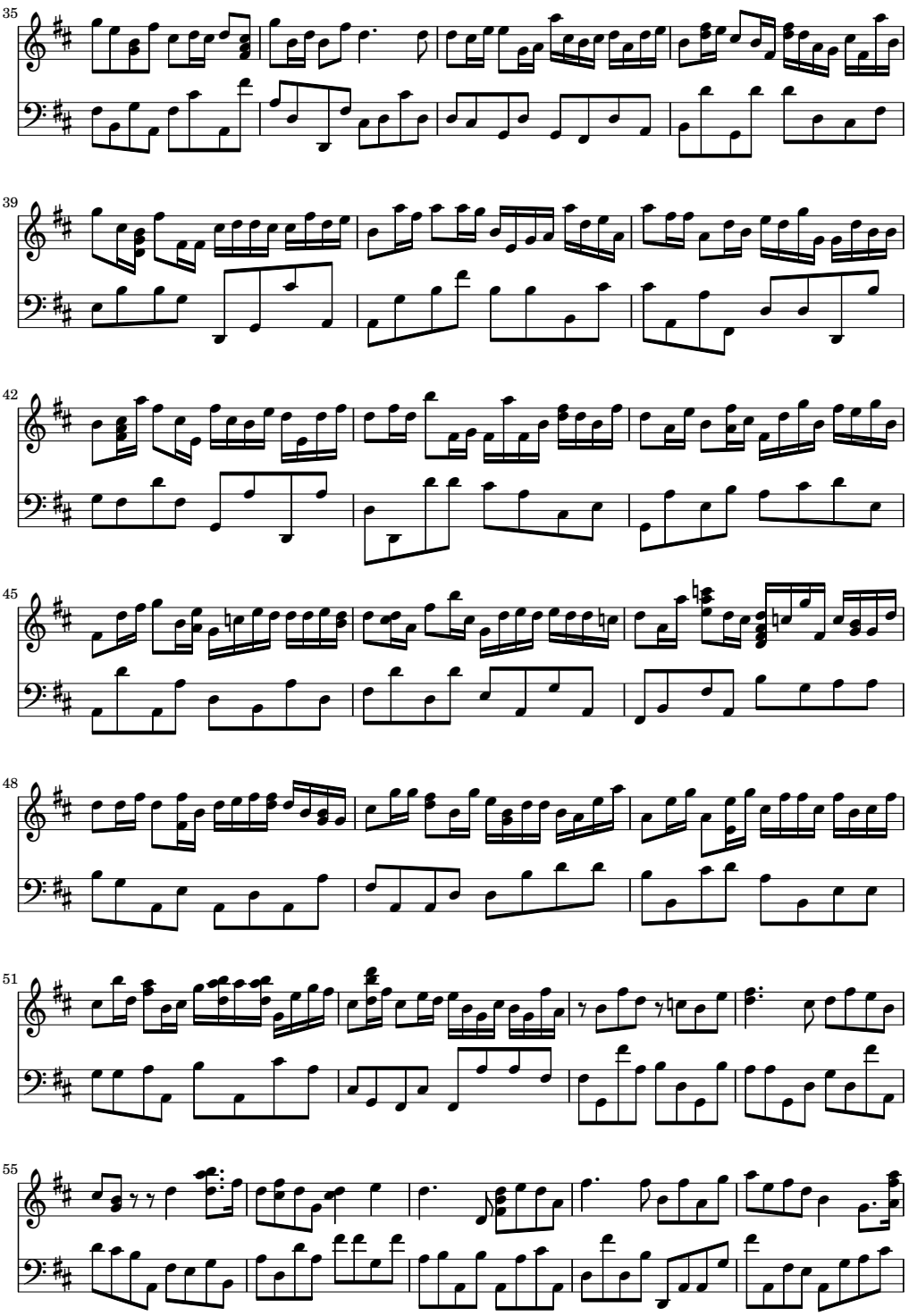


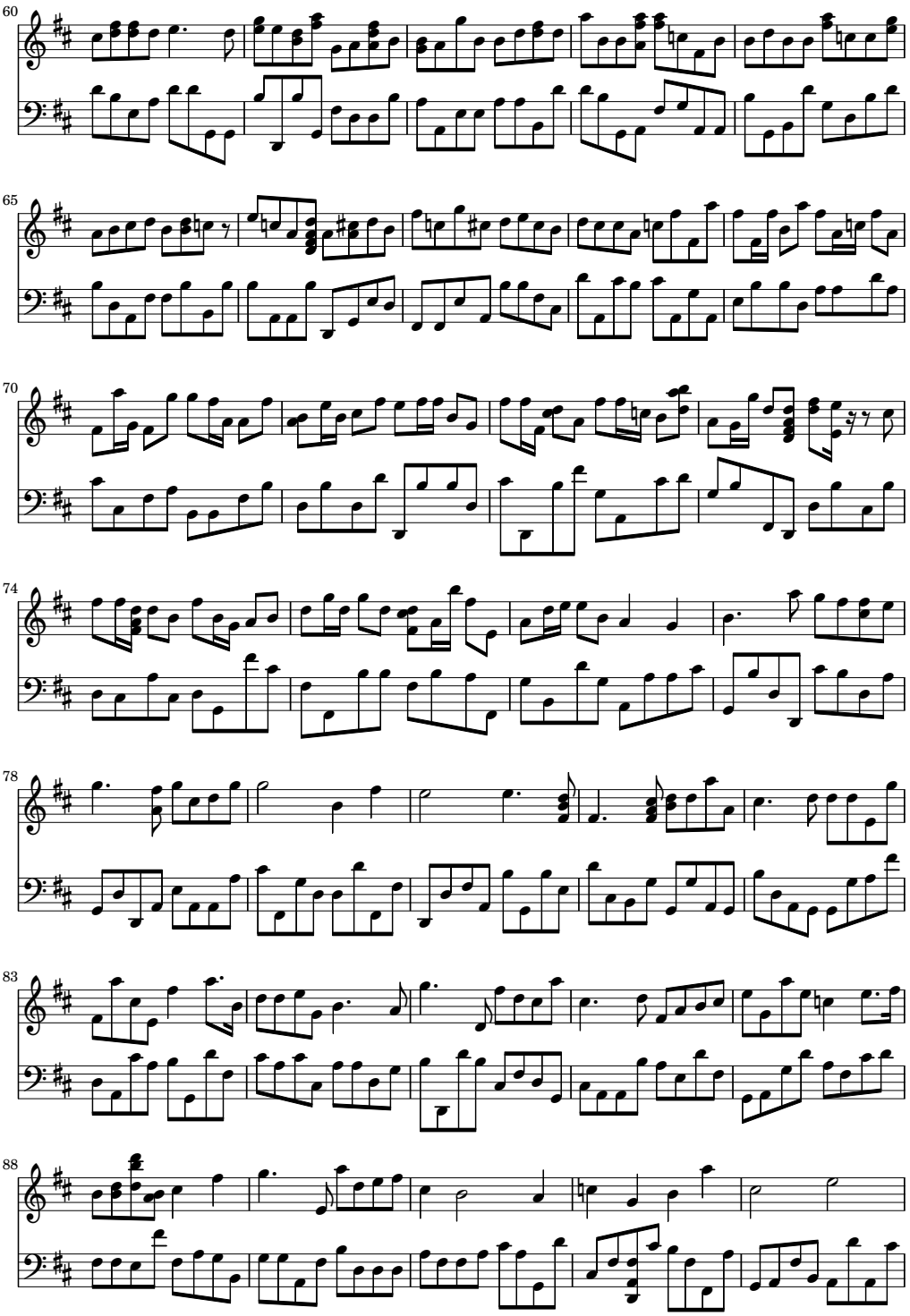

2
35
39
42
45
48
51
55


60
65
70
74
78
83
88

New Variaition from Mapping $\overleftarrow{\beta}$

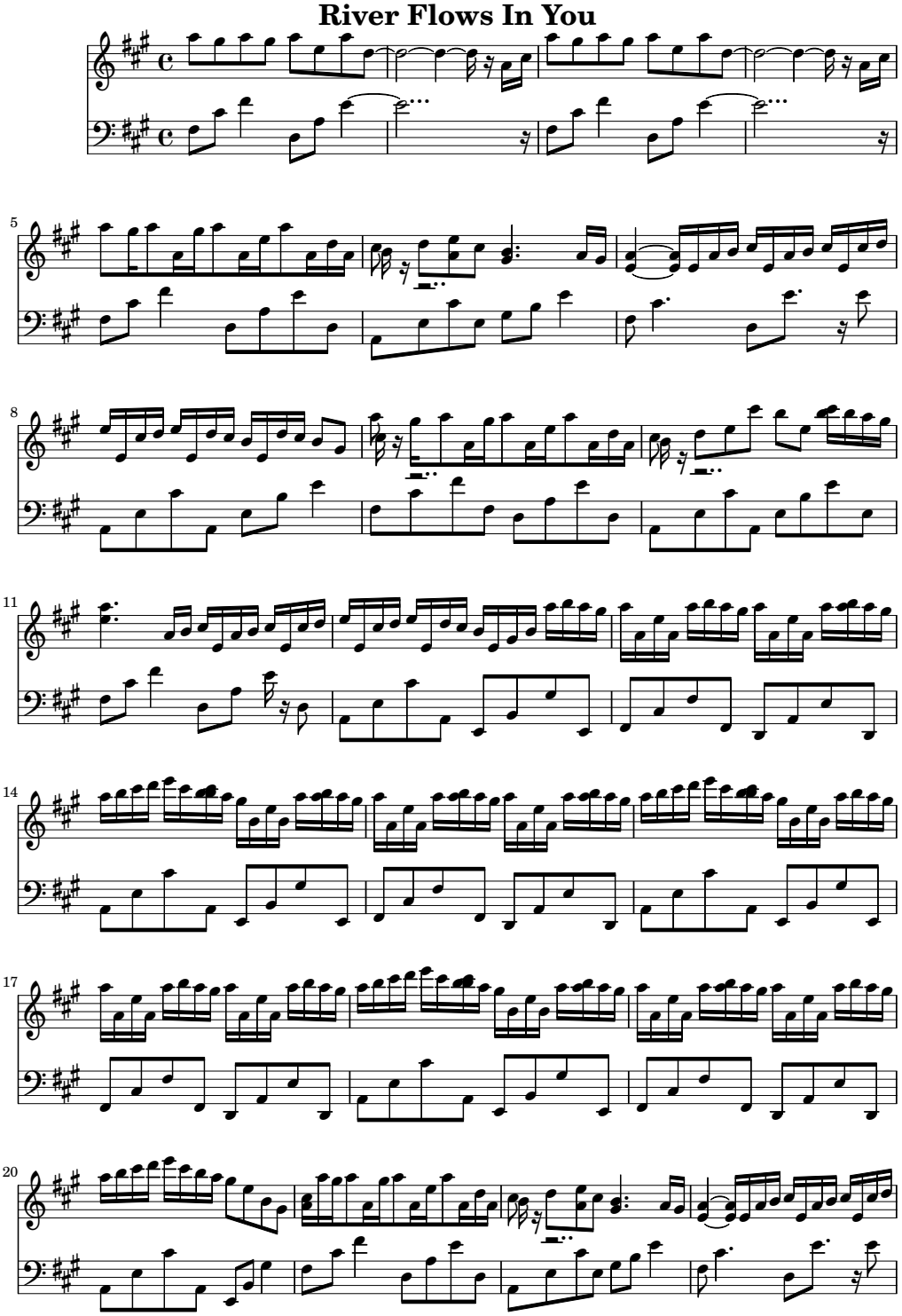


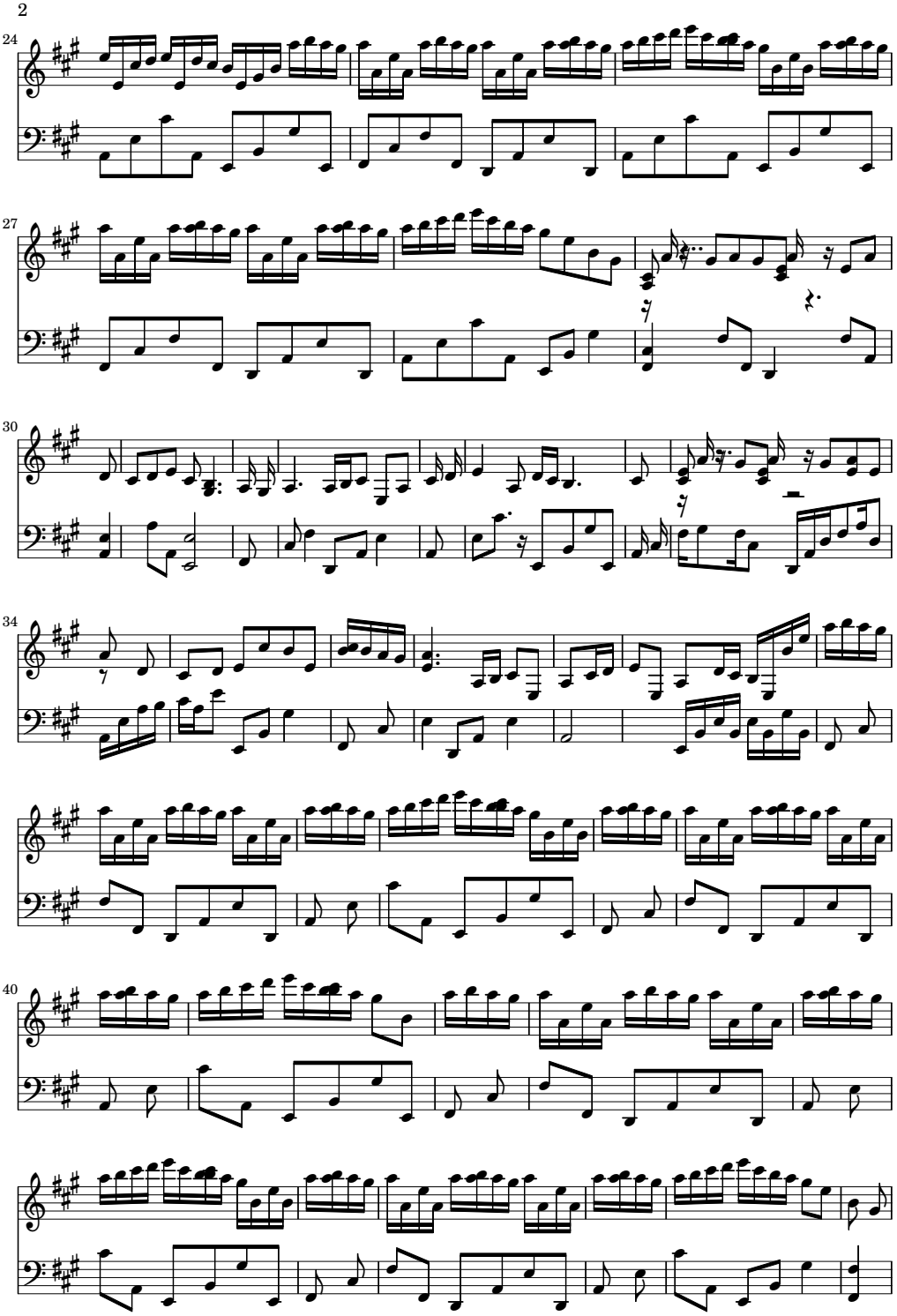

2
35
39
42
45
48
51
55

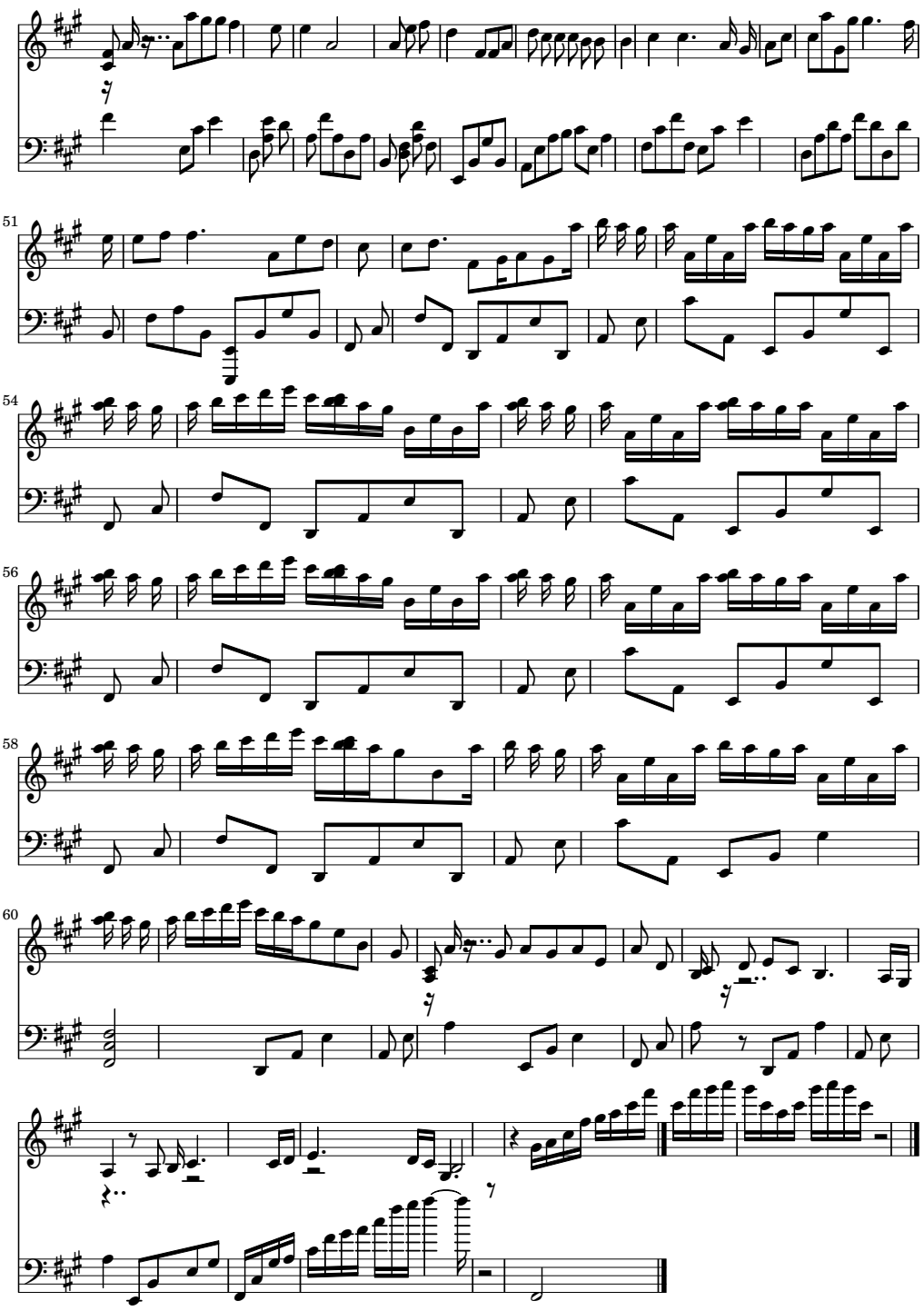

60
65
70
74
78
83
88

River Flows In You

Original Music Sheet

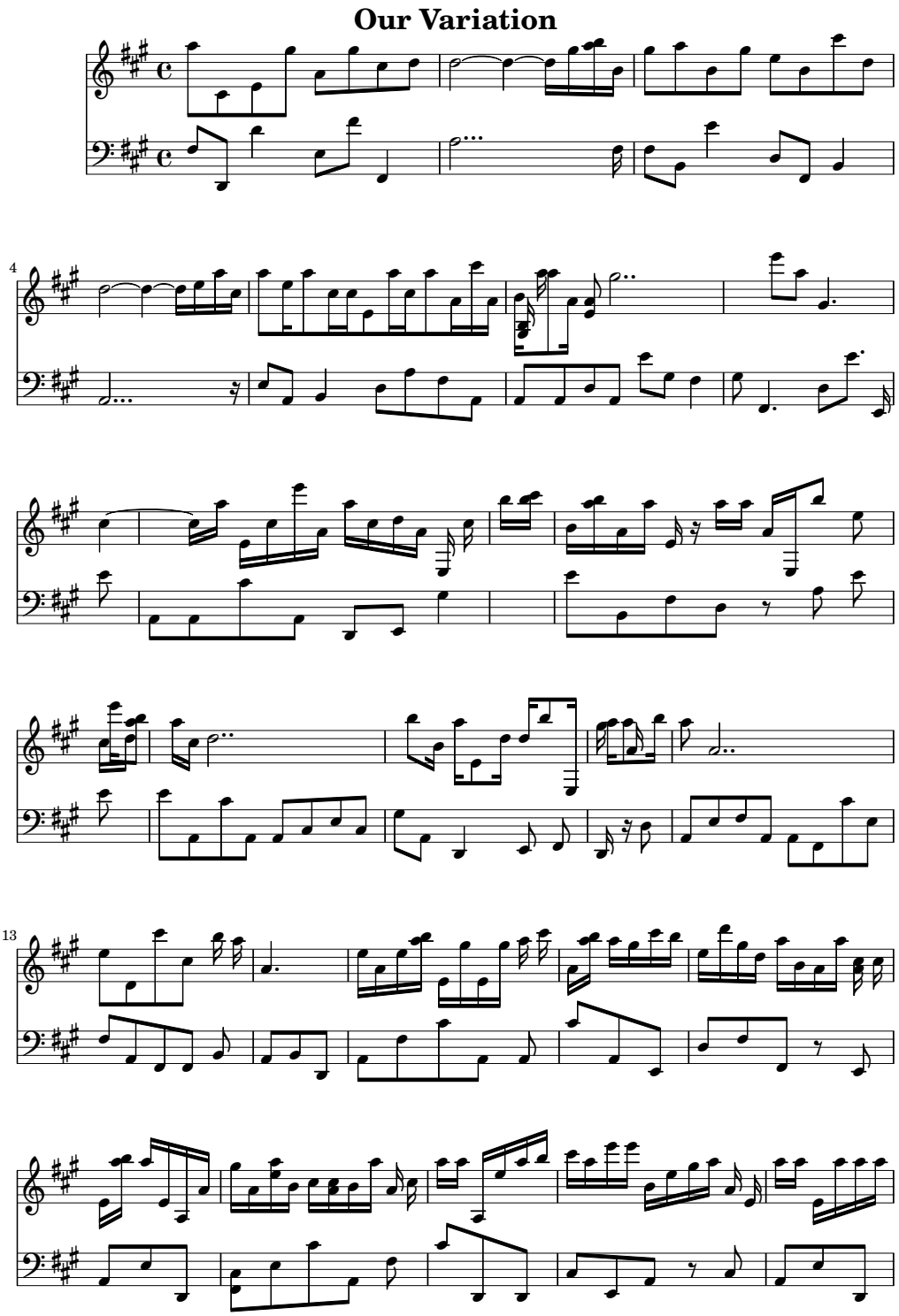


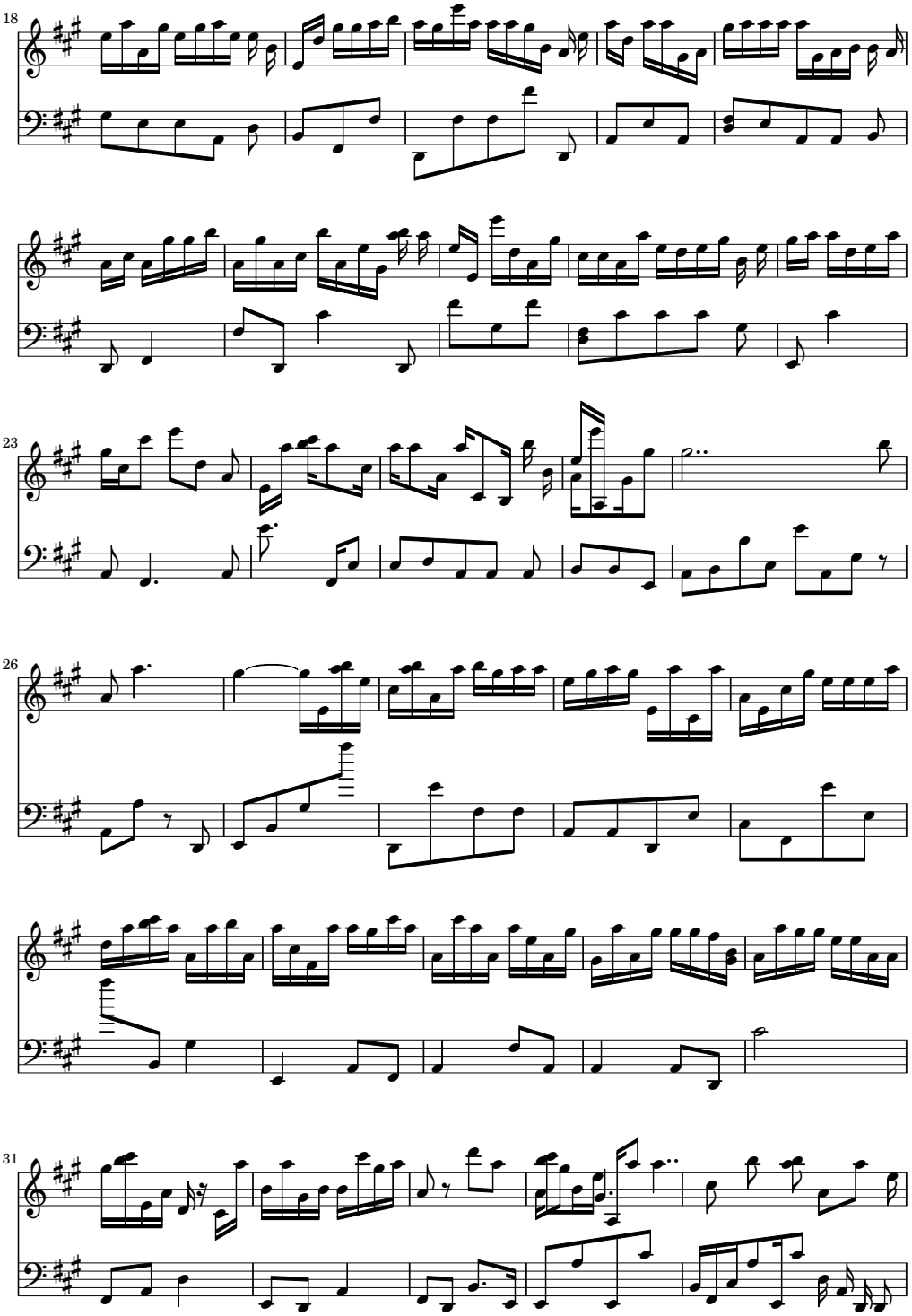

2
24
27
30
34
40

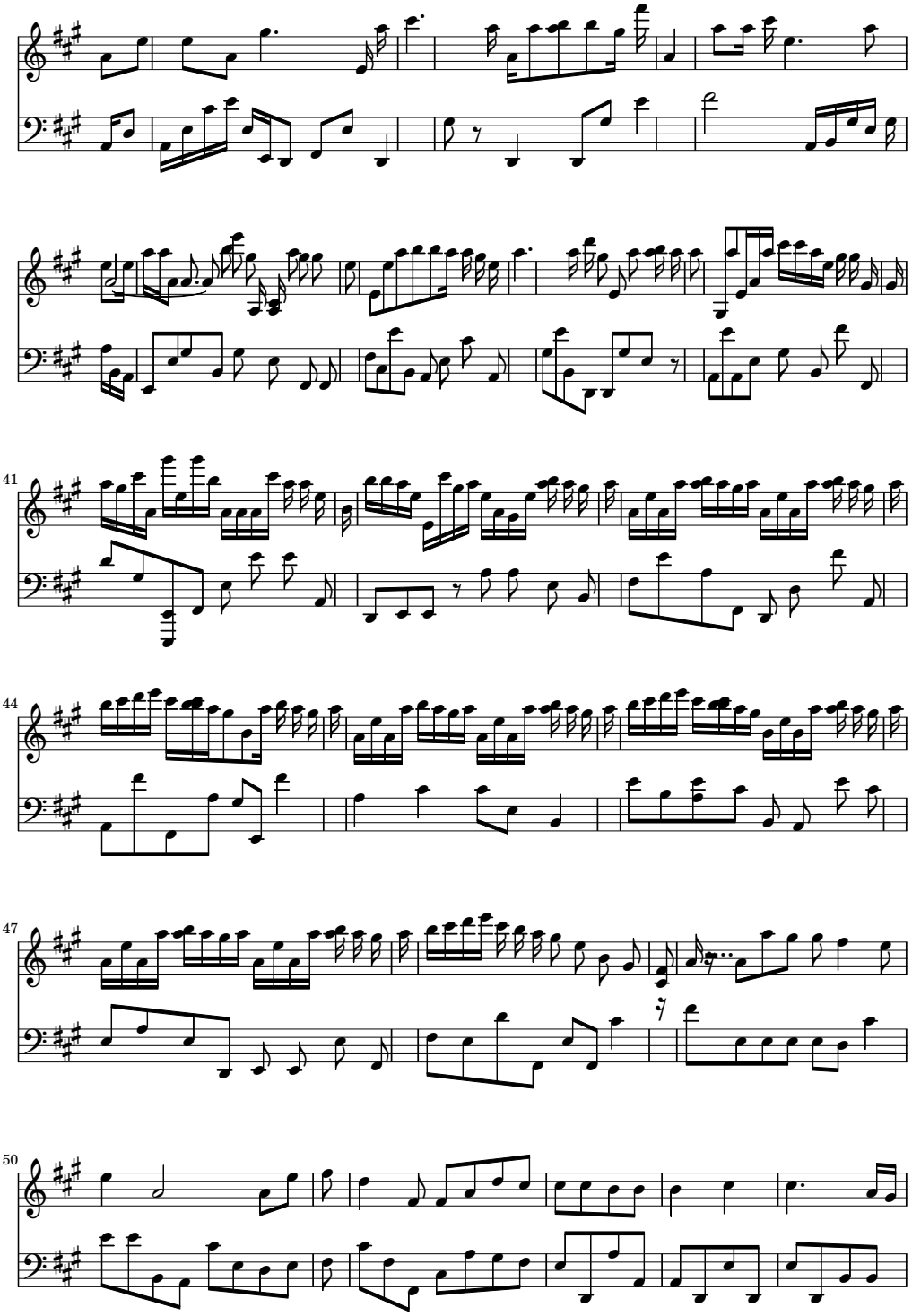


Music engraving by LilyPond 2.24.4—www.lilypond.org

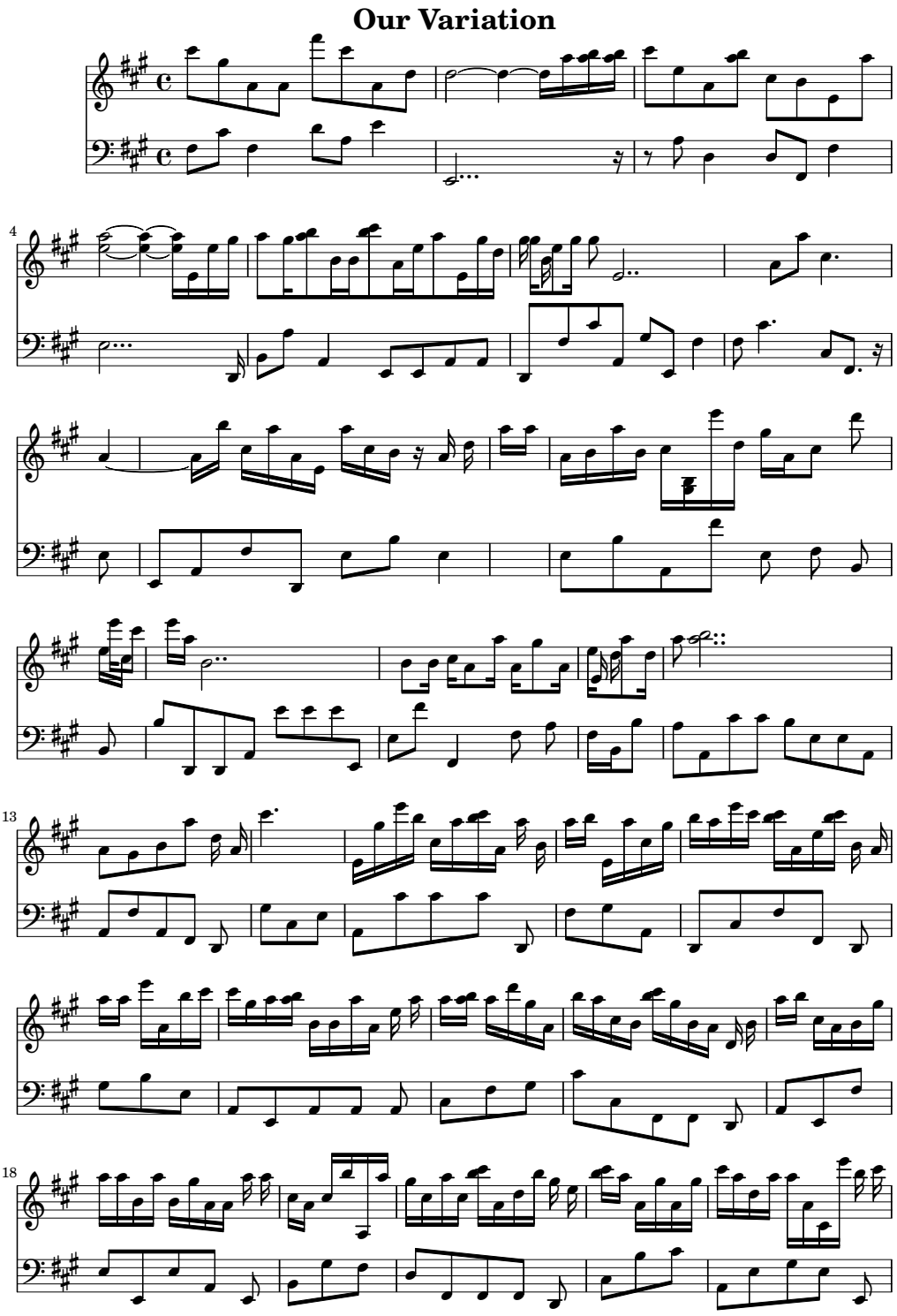
New Variaition from Mapping $\vec{\beta}$

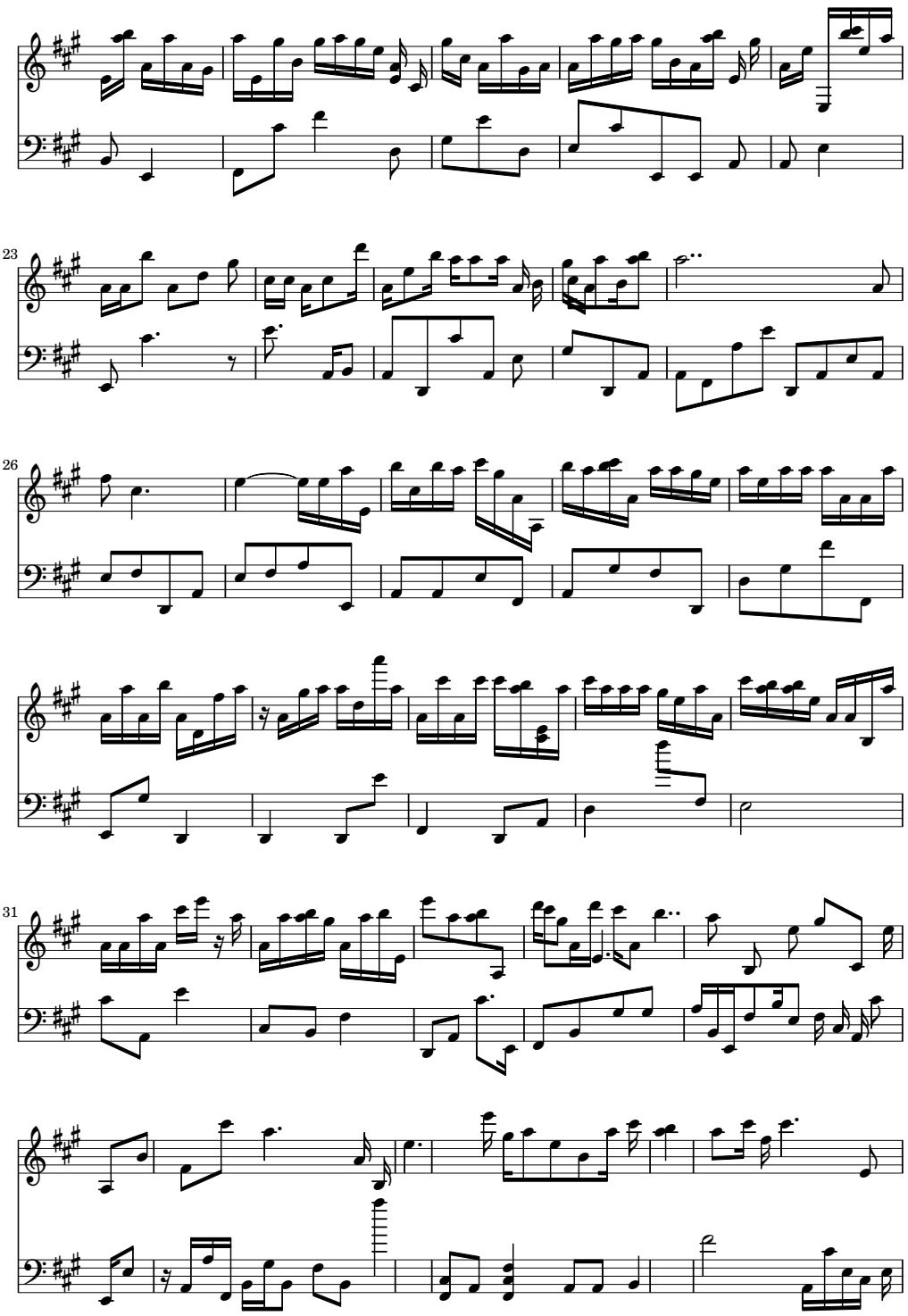



2
18
23
26
31



New Variaition from Mapping $\overleftarrow{\beta}$




2
23
26
31