**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ТАРАСА ШЕВЧЕНКА**
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

**Кваліфікаційна робота**
**на здобуття ступеня бакалавра**
за спеціальністю 122 Комп'ютерні науки

на тему:

# РОЗРОБКА ДОДАТКУ «АВТОТЮН»

# З ВИКОРИСТАННЯМ ПЕРЕТВОРЕННЯ ФУР'Є

Виконав студент 4-го курсу
Борисов Віталій Євгенович

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Кузенко Володимир Федорович

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«21» травня 2021 р.,
протокол № ___
Завідувач кафедри
М.С. Нікітченко

(підпис)

Київ – 2021

# РЕФЕРАТ

Обсяг роботи: 37 сторінок, 34 ілюстрації, 11 джерел посилань.

ЧАСТОТА, ОСНОВНА ЧАСТОТА, ГАРМОНІКА, СПЕКТР, ДИСКРЕТНЕ ПЕРЕТВОРЕННЯ ФУР'Є, ШВИДКЕ ПЕРЕТВОРЕННЯ ФУР’Є, ЗВОРОТНЕ ДИСКРЕТНЕ ПЕРЕТВОРЕННЯ ФУР’Є, НОТА, СИГНАЛ, СИНТЕЗ

Об’єктом роботи є розробка додатку «автотюн», що дозволяє корегувати висоту вхідного аудіо-сигналу.

Метою роботи є застосування засобів цифрової обробки сигналів для створення інструмента, що застосовуватиметься у музичній сфері.

Інструменти розроблення: PyCharm IDE.

Результати роботи: створено додаток, який змінює частоту коливань основної частоти і гармонік, що наявні в аудіо-сигналі.

# ЗМІСТ

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

**Автотюн –** пристрій або програма, що змінює частоту вхідного сигнала.

**Амплітуда коливань** – найбільше відхилення величини, яка періодично змінюється від деякого значення, умовно прийнятого за початкове.

**АЦП** – аналого-цифрове перетворення.

**Гармоніка –** хвиля, що має частоту, кратну частоті основної ноти.

**ДПФ** (дискретне перетворення Фур'є) – математична процедура, що визначає частотний (гармонійний) склад вхідного сигналу.

**Зворотне перетворення Фур'є** – математична процедура, що трансформує частотний спектр у початковий сигнал.

**Основна нота** – хвиля, частота коливання якої є найнижчою у спектрі.

**Періодична функція** – функція, що повторює свої значення з регулярними інтервалами.

**Спектр** – сукупність хвиль на всіх частотах, із яких складається сигнал.

**Частота** – кількість коливань за одиницю часу.

**Швидке перетворення Фур'є** – швидкий алгоритм перетворення Фур'є, що дозволяє виконати O(inputSize log inputSize) операцій замість $O(N^2)$.

**DAW (Digital Audio Workstation) -** програмний засіб для роботи зі звуком у музичній сфері.

**IDE** (integrated development environment, інтегроване середовище розробки) – комплексне програмне рішення для розробки програмного забезпечення.

**Python** – інтерпретована об'єктно-орієнтована мова програмування зі строгою динамічною типізацією.

**TWM –** two-way mismatch.

# ВСТУП

**Оцінка сучасного стану об’єкта розробки**. Сучасний світ неможливо уявити без музики. У світі композиторів та звукорежисерів існують певні тенденції, що призводять до покращення запису та подальшої роботи над музичними творами. Для досягнення «комерційного» звучання, усі елементи пісні повинні звучати дуже близько до ідеалу. Особливо це стосується домінуючої частини більшості сучасної музики – вокалу. Саме тому майже всі пісні, написані протягом останніх 10 – 15 років, записуються з використанням автотюну. Ця програма дозволяє корегувати висоту вокалу або інструментів.

Сьогодні на ринку присутні наступні додатки «автотюн»: Antares Audio Technologies Auto-Tune, Waves OVox, Melodyne, The Mouth та інші. Це точні та надійні інструменти, які обирають професіонали вже багато років.

**Актуальність роботи та підстави для її виконання.** Існуючі програми «автотюн» використовують методи цифрової обробки сигналів. Взагалі, цифрова обробка сигналів застосовуються під час роботи не тільки з аудіо, а ще й відео, зображеннями, прогнозами погоди, астрономічними та економічними процесами тощо.

Застосування математичних операцій та комп’ютерних технологій у музиці, мої особиста зацікавленість у цих сферах та бажання дізнатися, як влаштована робота з сигналами, стали підставами для виконання даної роботи.

**Мета й завдання роботи.** Метою цієї роботи є розробка додатку, який буде отримувати на вхід аудіо-сигнал та змінювати його висоту. Завданням є отримання спектру частот вхідного сигнала, знаходження основних нот і гармонік, побудова стохастичної та гармонійної моделей з новими частотами та їх конвертація у вихідний сигнал.

**Об'єкт, методи й засоби розроблення.** Для розв’язання поставленої задачі було розроблено додаток, що реалізовує необхідні алгоритми. Мова розробки – Python, середовище розробки – PyCharm IDE.

# РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Будь-який звук є коливанням середовища. Зазвичай під середовищем мається на увазі повітря. У цій роботі розглядаються періодичні коливання – коливання, описані функціями, що повторюють свої значення з регулярними інтервалами[1]. Як і будь-яке коливання, коливання повітря описується рівнянням,

$$x = \cos(\omega t + \varphi_0) \qquad (1.1)$$

де $\omega$ – частота коливань, $\varphi_0$ – початкове положення.

У музиці коливання з певними частотами називають нотами. Проте, коли ми чуємо ноту, зіграну на будь-якому інструменті, ми чуємо не лише коливання з частотою, що відповідає якійсь ноті, а ще й багато гармонік. Тобто, звук складається з основної ноти та гармонік. Сукупність усіх коливань повітря, що створює будь-яке джерело звуку, називають спектром частот. Основна нота – це коливання з найнижчою частотою у спектрі, гармоніки – це коливання із частотами, що є кратними частоті основної ноти (див. рис. 1.2). Саме тому звук – це функція , що є сумою синусоїдальних функцій (див. рис. 1.3 - 1.4).

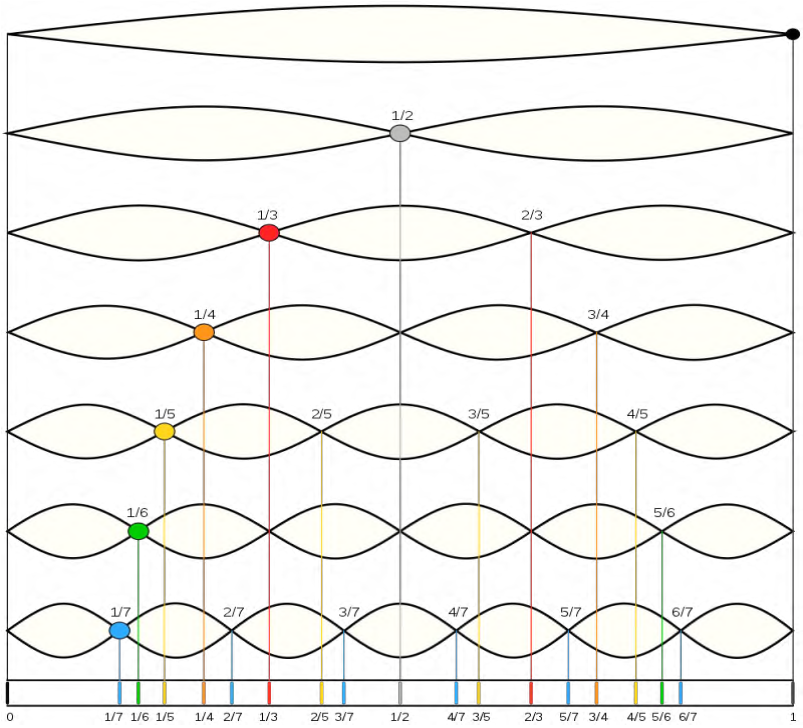


Рисунок 1.2 - гармоніки

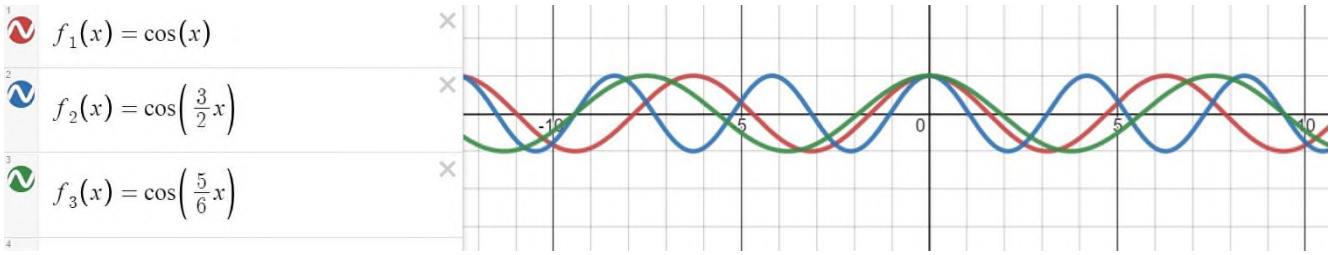


Рисунок 1.3 – графік гармонік

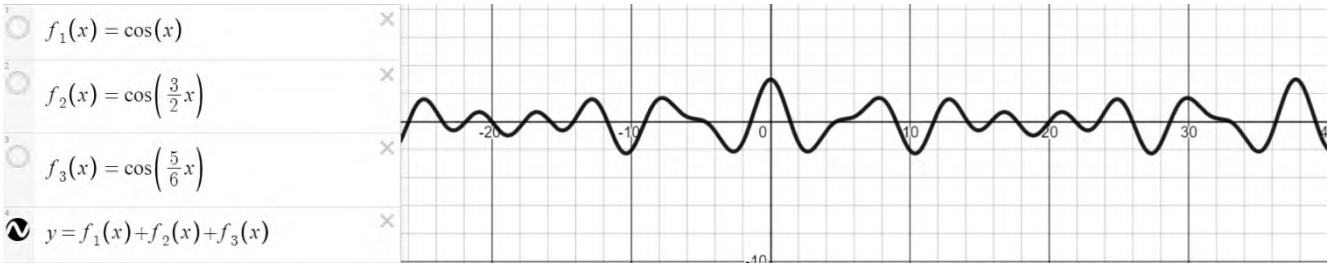


Рисунок 1.4 – графік суми гармонік

Слід зазначити, що кожний тип музичних інструментів має характерний для нього частотний спектр. Наприклад, одна нота, зіграна на фортепіано, змушує відповідну струну коливатись із частотою основної ноти та багатьма іншими частотами одночасно. До звуку також додаються вібрація деревини, із якої виготовлений інструмент, та коливання сусідніх струн. Кожна гармоніка, що присутня у цьому спектрі, має свою частоту й амплітуду (гучність) коливань. Наявність певних гармонік та амплітуда кожної з них формують характерне, впізнаване звучання інструмента (тембр). На рисунку 1.5 зображені гармоніки, присутні у звуці лише однієї ноти Ля, що має частоту 440 Гц. Червоним позначені основна нота і її частота, жовтим і зеленим – друга (октава) і третя гармоніки та їх частоти відповідно.


Рисунок 1.5 – гармоніки фортепіано

Підсумовуючи вище сказане, отримуємо наступний висновок: звук будь-якого музичного інструмента є коливанням повітря, що описується складною синусоїдальною функцією (сумою простих синусоїдальних функцій). Тому можна більш точно сформулювати завдання, яке розв’язується у роботі:

1. Отримати із вхідного аудіо-сигнала спектр частот (рис.1.6);
2. Знайти основні ноти;
3. Побудувати гармонійно-стохастичну модель на заданій користувачем новій частоті;

4. Синтезувати побудовані моделі у вихідний сигнал.

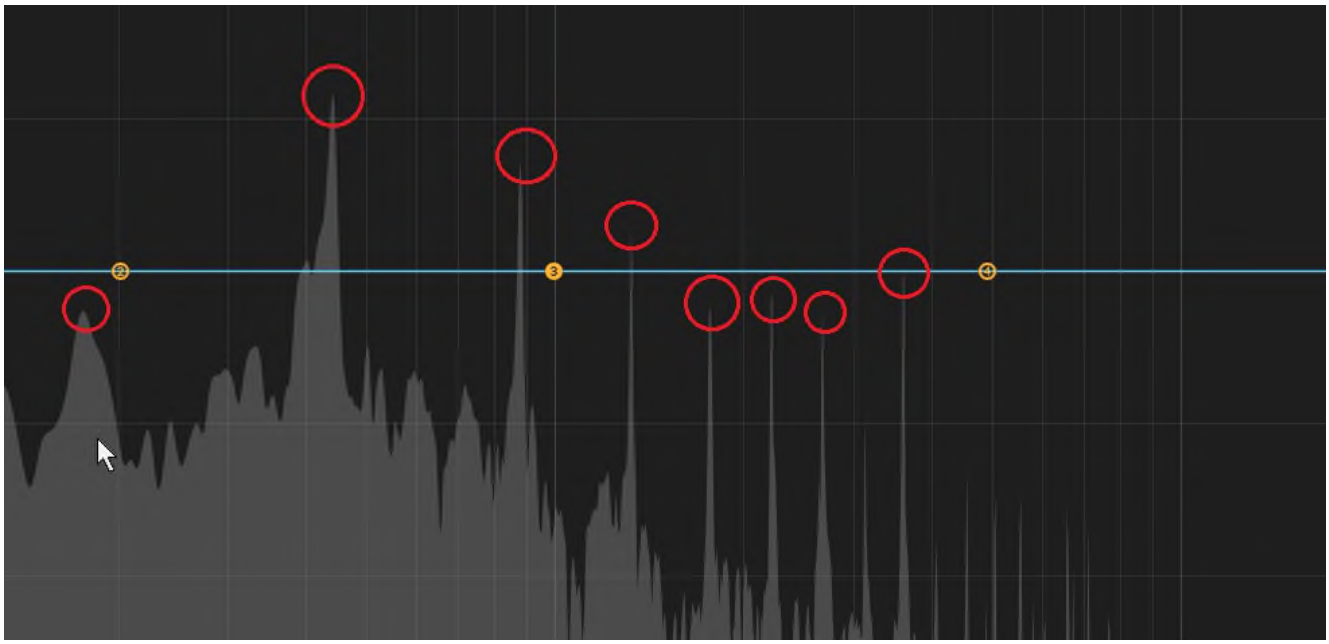

Рисунок 1.6 – піки у спектрі

Усі звуки, які розпізнаються людським вухом, є аналоговими сигналами. Сигнал – функція, що передає інформацію про якесь фізичне явище[2]. Аналоговий сигнал – будь-який неперервний сигнал[3]. Але комп'ютери не мають можливості працювати з аналоговими сигналами. Для того, щоб передати комп'ютеру сигнал, необхідно отримати цифровий сигнал за допомогою аналого-цифрового перетворення. Цифровий сигнал – це послідовність дискретних значень[4]. Аналого-цифрове перетворення – це перетворення аналогового сигнала у цифровий. Суть АЦП полягає у наступному: на вхід пристрою, що виконує АЦП, подається аналоговий сигнал, що є неперервною функцією. Пристрій реєструє значення цієї функції через рівні проміжки часу та формує дискретну функцію (див. рис. 1.7). Кількість таких реєстрацій за одиницю часу називають частотою дискретизації.

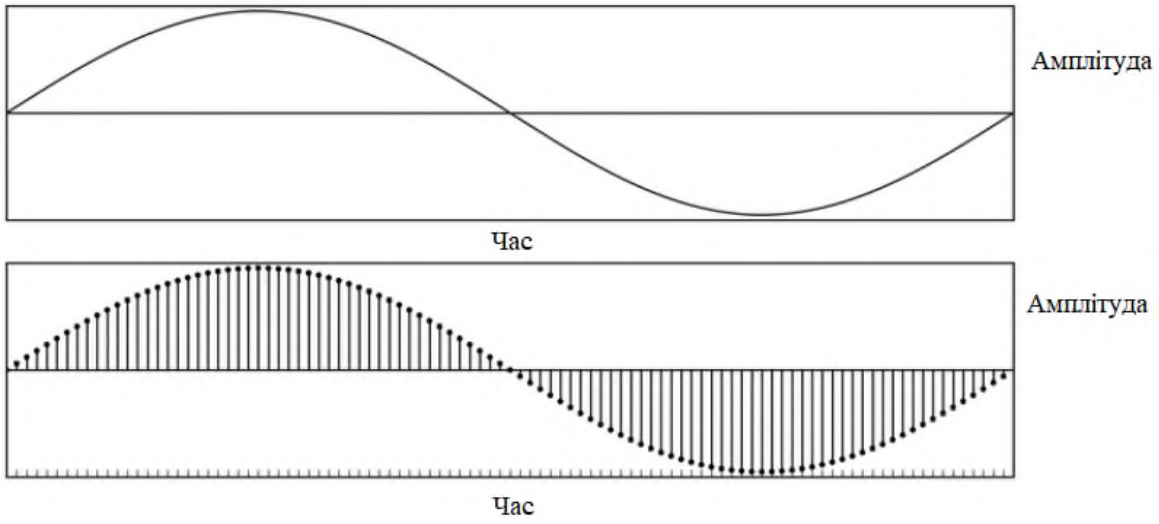


Рисунок 1.7 – аналого-цифрове перетворення

# РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Для візуалізації спектрів на рис. 1.5 та 1.6 була використана DAW під назвою “Ableton Live”. Це один із найпопулярніших програмних засобів для роботи зі звуковими файлами (рис. 2.1).

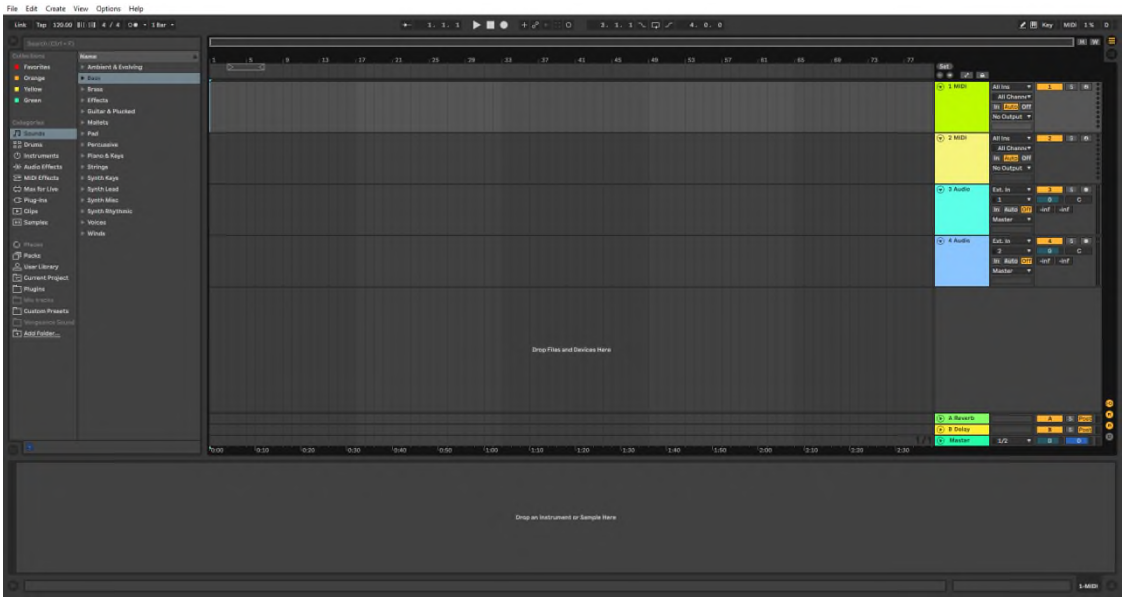

Рисунок 2.1 – Ableton Live

Як і численні інші DAW, Ableton Live має великий набір різноманітних інструментів, серед яких – еквалайзер (рис. 2.2).

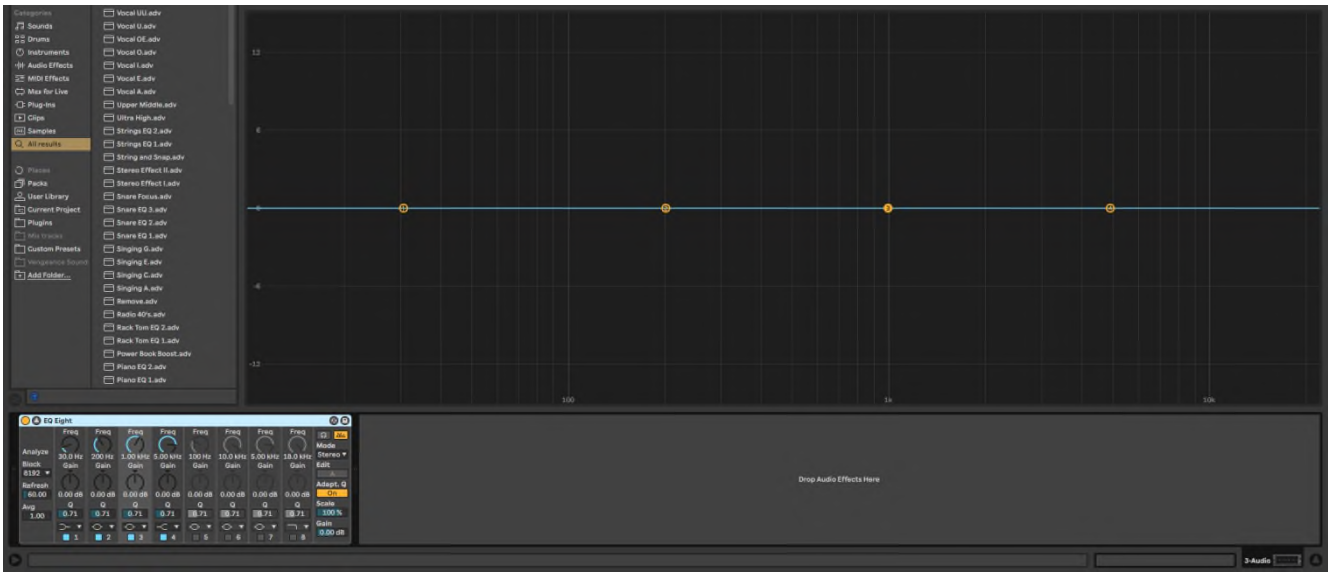

Рисунок 2.2 – еквалайзер Ableton Live

Еквалайзер – це пристрій або програмний за стосунок, який використовується для налаштування частотного балансу вхідного сигнала[5]. Іншими словами, еквалайзер дозволяє збільшити або зменшити гучність частот, наявних у сигналі, та в деяких випадках візуалізувати частотний спектр (рис. 2.3 – 2.4).

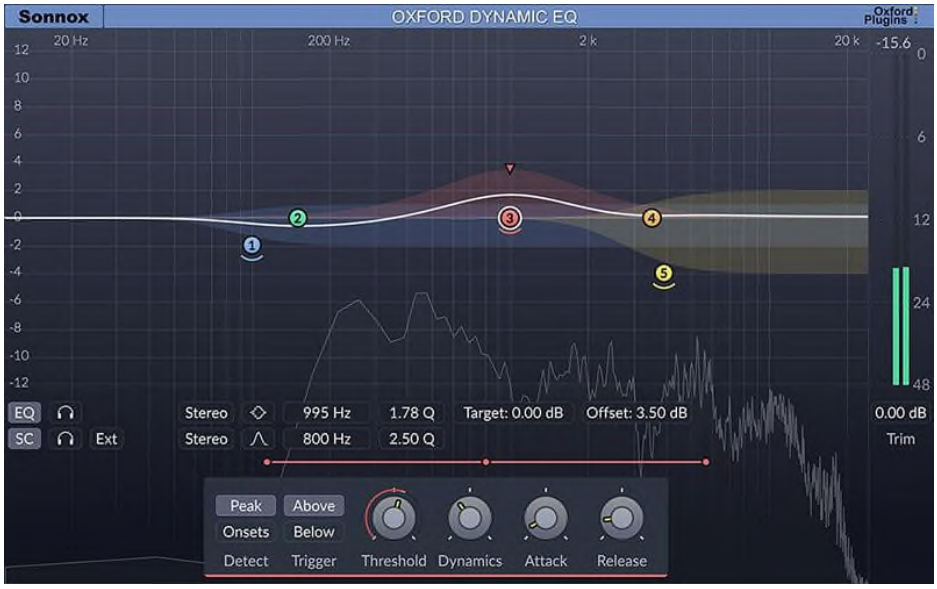


Рисунок 2.3 – еквалайзер із візуалізацією спектра


Рисунок 2.4 – еквалайзер без візуалізації спектра

У якості мови для розробки системи було обрано Pyhon, у якості IDE – PyCharm IDE.

# РОЗДІЛ 3. ДИСКРЕТНЕ ПЕРЕТВОРЕННЯ ФУР'Є

В основі даної роботи лежить ДПФ. ДПФ – доволі потужний інструмент для роботи з сигналами будь-якої природи. Він застосовується у роботі з аудіо- та відео-файлами, зображеннями, в астрономії, при прогнозуванні погоди та економіки тощо.

ДПФ дозволяє виокремити всі періодично повторювані явища при дослідженні якогось процесу. Дуже гарним прикладом цього є пошук нових планет за допомогою дослідження яскравості зірки. Якщо вимірювати яскравість зірки, навколо якої обертається невідома планета, протягом певного часу, можна зареєструвати зміну яскравості: коли небесне тіло потрапляє між необхідним приладом і спостережуваною зіркою, частина світла, що випромінюється останньою, не доходить до приладу. Через це реєструється менша яскравість, ніж зазвичай (рис. 3.1). Після цього планета перестає перешкоджати світлу, тому яскравість зірки повертається до попереднього значення. У цьому випадку маємо дискретну функцію, що описує яскравість зірки, яка змінюється періодично. Тому, застосувавши ДПФ до цієї функції, можна знайти частоту обертань ще невідомої планети навколо зірки, а, отже, радіус, масу та інші параметри планети.

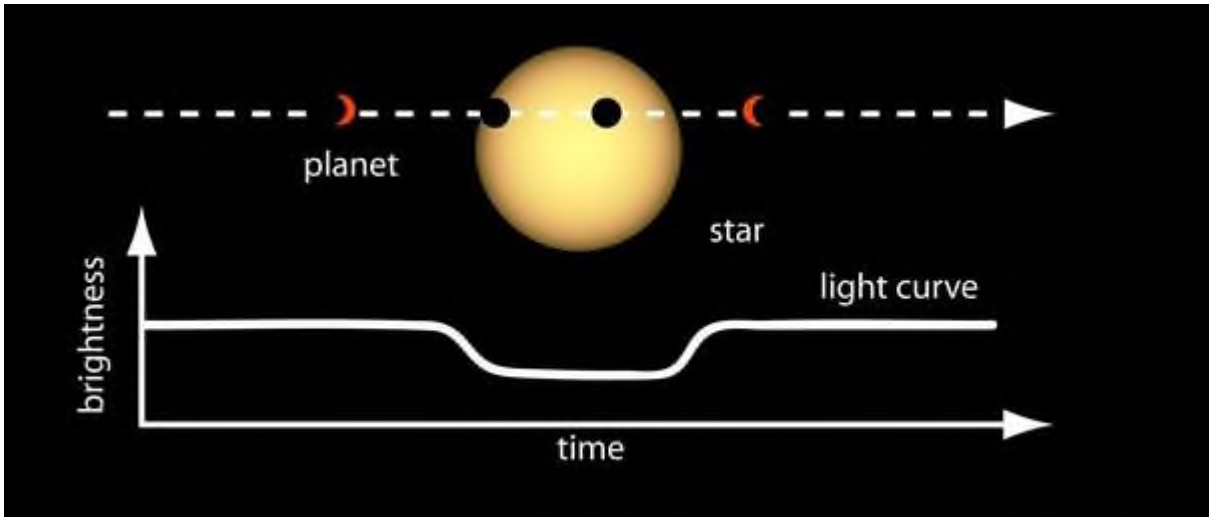


Рисунок 3.1 – зміна яскравості зірки через планету, що обертається навколо неї

Отже, дискретне перетворення Фур'є описується наступною формулою:

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{\frac{-2\pi i}{N}kn}, n \in \mathbb{Z} \quad (3.1)$$

У цьому випадку $x[n]$ – дискретна функція, що описує сигнал; $N$ – кількість семплів (значень, що були зареєстровані під час аналого-цифрового перетворення. Це значення можна отримати, помноживши частоту дискретизації на тривалість сигнала у секундах); $k$ – індекс частоти; $X[k]$ – масив, індексами якого є індекси частот, а значення – амплітудами коливань із відповідними частотами[6].

За допомогою формули Ейлера (формула 3.2) можна представити ДПФ у наступному вигляді (формула 3.3):

$$e^{ix} = \cos x + i \sin x \quad (3.2)$$

$$X[k] = \sum_{n=0}^{N-1} x[n][\cos\left(\frac{2\pi kn}{N}\right) + i \sin(\frac{2\pi kn}{N})], \quad k = (0, \dots, N-1) \quad (3.3)$$

На рисунку 3.2 зображений приклад візуалізації спектра $X[k]$. Для цього прикладу було синтезовано сигнал, який складається з трьох гармонік, що формують мінорний тризвук. Слід зазначити, що через природу ДПФ у спектрі присутні гармоніки з протилежними індексами. Тому в подальшому аналізі беруть участь лише гармоніки з додатними індексами.

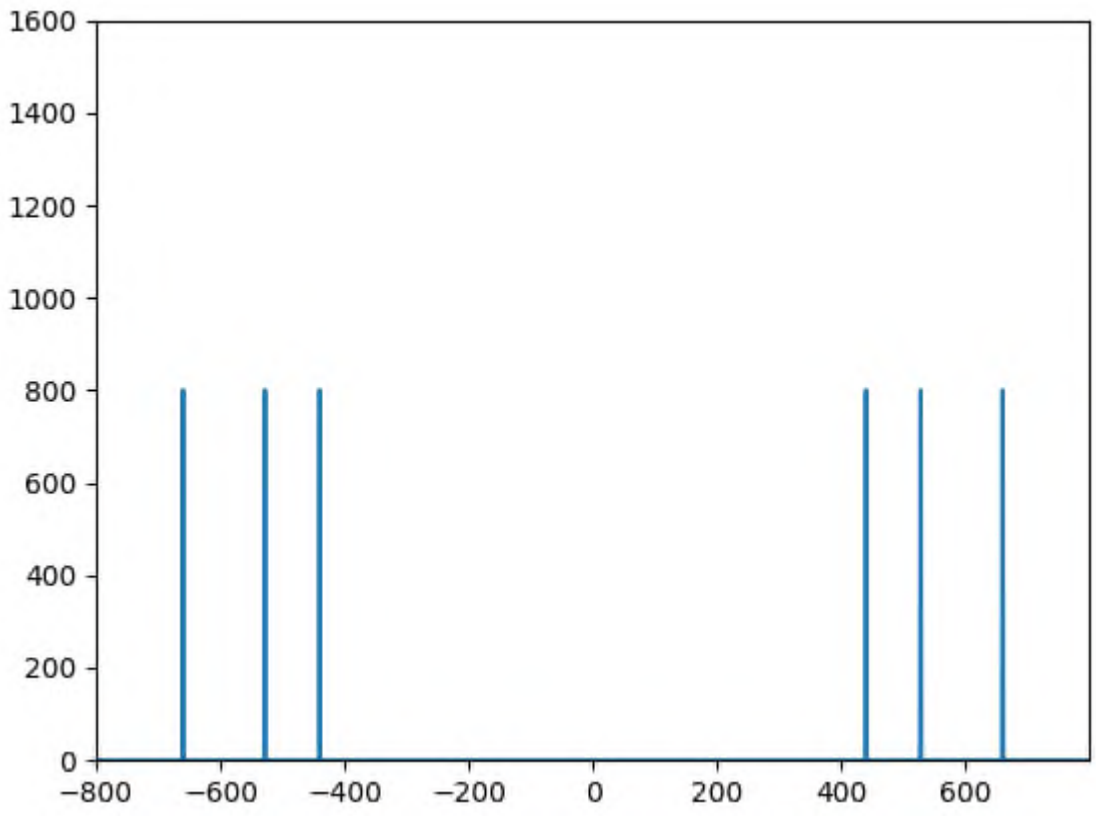


Рисунок 3.2 – графік спектра

Дискретне перетворення Фур’є також можна подати у матричному вигляді:

$$\hat{A} = \begin{pmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & e^{-\frac{2\pi i}{N}} & e^{-\frac{4\pi i}{N}} & e^{-\frac{6\pi i}{N}} & \dots & e^{-\frac{2\pi i}{N}(N-1)} \\ 1 & e^{-\frac{4\pi i}{N}} & e^{-\frac{8\pi i}{N}} & e^{-\frac{12\pi i}{N}} & \dots & e^{-\frac{2\pi i}{N}2(N-1)} \\ 1 & e^{-\frac{6\pi i}{N}} & e^{-\frac{12\pi i}{N}} & e^{-\frac{18\pi i}{N}} & \dots & e^{-\frac{2\pi i}{N}3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-\frac{2\pi i}{N}(N-1)} & e^{-\frac{2\pi i}{N}2(N-1)} & e^{-\frac{2\pi i}{N}3(N-1)} & \dots & e^{-\frac{2\pi i}{N}(N-1)^2} \end{pmatrix} \quad (3.4)$$

У цьому випадку елементи матраці A обчислюються за формулою

$$A(m,n) = \exp(-2\pi \frac{(m-1)(n-1)}{N}) \quad (3.5),$$

Тоді спектр можна обчислити за формулою

$$\overline{X} = A\overline{x} \quad (3.6)$$

Матрична форма дискретного перетворення Фур’є дає змогу оптимізувати обчислення та використовувати швидке перетворення Фур’є, що буде описане у наступному розділі.

# РОЗДІЛ 4. ШВИДКЕ ПЕРЕТВОРЕННЯ ФУР’Є

Як відомо, дискретне перетворення Фур’є перетворює вхідний сигнал у представлення у частотному домені, тобто подає інформацію про сигнал в залежності від частоти замість часу[8]. ДПФ отримується шляхом розкладання послідовності значень на компоненти різних частот.

ДПФ – операція, корисна в багатьох сферах, але її обчислення безпосередньо за визначенням часто є надто повільним, щоб бути практичним. Швидке перетворення Фур’є обчислює такі перетворення із помітно більшою швидкістю, розкладаючи матрицю ДПФ на добуток матриць, що значно спрощують подальші обчислення. Як результат, вдається зменшити складність обчислення ДПФ $O(N^2)$, що виникає при звичайному застосуванні ДПФ, до O(Nlog(inputSize)), де inputSize – кількість семплів у сигналі. Різниця в швидкості може бути величезною, особливо для великих наборів даних, де inputSize досягає тисяч або мільйонів. За наявності помилки округлення багато алгоритмів швидкого перетворення Фур’є набагато точніші, ніж оцінка визначення ДПФ.

Існує багато різних алгоритмів швидкого перетворення Фур’є, заснованих на широкому діапазоні опублікованих теорій, від простої арифметики складних чисел до теорії груп та теорії чисел.

Швидкі перетворення Фур'є широко використовуються для застосування в техніці, музиці, науці та математиці. Основні ідеї були популяризовані в 1965 р., але деякі алгоритми були виведені ще в 1805 р. У 1994 р. Гілберт Стренг описав швидке перетворення Фур’є як "найважливіший числовий алгоритм у нашому житті", і його включили до 10 найкращих алгоритмів 20 століття у журналі IEEE "Обчислення в науці та техніці"[9].

Розкладання матриці ДПФ на більш прості матриці виконується за формулою:

$$\overline{X} = F_{2N}\overline{f} = \begin{bmatrix} I_N & -D_N \\ I_N & -D_N \end{bmatrix} \begin{bmatrix} F_N & 0 \\ 0 & F_N \end{bmatrix} \begin{bmatrix} f_{even} \\ f_{odd} \end{bmatrix} \quad (4.1)$$

У цьому випадку $I_N$ – матриця ідентичності розмірності inputSize x inputSize; $D_N$ – діагональна матриця тієї ж розмірності; $\begin{bmatrix} f_{even} \\ f_{odd} \end{bmatrix}$ – вектор вхідних даних, реорганізований наступним чином: верхня частина – значення вектора з парними індексами, нижня – значення вектора з непарними індексами.

Такі операції з матрицями можна продовжувати до тих пір, доки розбиття початкової матриці матиме розмірність 2x2.

Важливим є наступний момент: даний алгоритм можна застосувати до сигнала, кількість семплів у якому є степенем 2. Але якщо розмірність вектора не є степенем 2, вхідні дані можна доповнити 0 до найближчого степеня 2. Навіть при такому доповненні даних швидке перетворення Фур’є працюватиме набагато швидше, ніж ДПФ.

# РОЗДІЛ 5. ЗНАХОДЖЕННЯ ФУНДАМЕНТАЛЬНОЇ ЧАСТОТИ

Сьогодні існує багато методів оцінки основної (фундаментальної) частоти $F_0$. Серед методів, що працюють із часовим представленням сигнала (залежності значення функції від часу) можна виділити наступні:

- мультиплікативна автокореляція;
- субтрактивна автокореляція (функція середньої різниці величин);
- методи, засновані на лінійному прогнозуванні.

До методів, що працюють із частотним представленням сигнала, належать наступні:

- кепструм;
- гістограма періоду;
- методи максимальної ймовірності;
- процедури співставлення гармонійних пат тернів.

Постійний дослідницький інтерес у цій галузі свідчить про те, що досі не існує жодної повністю успішної системи знаходження основних частот для широкого кола типів звукових сигналів. Без повного об’єктивного порівняння різних процедур знаходження $F_0$ важко зробити конкретні висновки про сильні та слабкі сторони існуючих методів. Тим не менш, алгоритми, які прямо не враховують наявність шуму, реверберації та інших поширених деградацій сигнала, особливо складно оцінити для цілей реального аналізу. Підходи в часовому представленні використовують передбачувану періодичну природу вхідного сигналу шляхом виявлення таких характеристик сигналу, як піки, нульові переходи та інші періодичні структури. Очікується, що тривалістю часу між повторюваними відповідними ознаками буде період сигнала, що визначається як $1/ F_0$. Інші методи часового домену використовують автокореляційний підхід для ідентифікації періодів форми сигналу, заснований на уявленні, що ми очікуємо, що один цикл періодичного сигналу буде сильно корелювати з наступним.

Подібним чином методи частотного представлення використовують той факт, що спектри періодичних сигналів часового домену демонструють квазігармонічні спектральні структури, що проявляються регулярно розташованими піками в спектрі величин. Таким чином, проблема оцінки частоти стає завданням визначення набору гармонійних частот, які, в якомусь сенсі, найкраще відповідають положенням спектральних піків.

У цій роботі для знаходження фундаментальної частоти було використано алгоритм TWM (двовимірна невідповідність). Процедура двосторонньої невідповідності нагадує оцінку максимальної ймовірності, оскільки вимірюваний спектр порівнюється з постульованою гармонічною спектральною картиною. Процедура оцінки TWM базується на порівнянні кожної виміряної послідовності гармонік із аналізу ДПФ із передбаченими послідовностями гармонік на основі випробувальних значень $F_0$. Невідповідність між виміряною та передбачуваною гармоніками називається помилкою невідповідності. Звичайно, помилка невідповідності була б нульовою, якби передбачене значення $F_0$ точно відповідало фактичному фундаментальному, а вимірюваний спектр складався виключно з кратних гармонік. Крім того, в реальних ситуаціях, коли присутні шум і похибка вимірювань, помилка невідповідності ніколи не буде рівно нульовою.

Розглянемо приклад послідовності частинок: 200, 300, 500, 600, 700, 800 Гц. Вибір $F_0$ = 100 Гц дасть передбачувану послідовність 100, 200, 300, 400, 500, 600, 700, 800 Гц, де передбачені компоненти при 100 та 400 Гц не знаходяться у заданій послідовності, але інші вимірювані компоненти враховуються. Вибір $F_0$ = 50 Гц також повністю охоплює виміряні часткові показники, але багато передбачуваних гармонік (50, 100, 150, 250, 350, 400, 550 Гц та інші) не знаходяться у заданій послідовності. Аналогічним чином, вибір $F_0$ = 200 Гц призводить до передбачуваної послідовності гармонік: 200, 400, 600, 800 Гц, яка правильно прогнозує деякі виміряні частки, але пропускає інші. Тому необхідні певні засоби виявлення найкращої відповідності між передбаченими та виміряними гармоніками.

Метод двосторонньої невідповідності полягає у знаходженні двох похибок невідповідності. Перший розрахунок базується на різниці частот між кожною гармонікою у заданій послідовності та найближчою до неї гармонікою у передбачуваній послідовності. Другий розрахунок заснований на невідповідності між кожною гармонікою у передбачуваній послідовності та найближчою до неї гармонікою у заданій послідовності (рис.5.1). Таке двостороннє обчислення допомагає уникнути октавних помилок, застосовуючи «покарання» за гармоніки, які є у заданій послідовності гармонік, але не є у прогнозованій, а також для гармонік, присутність яких у послідовності передбачена, але насправді не відображається у заданій послідовності.

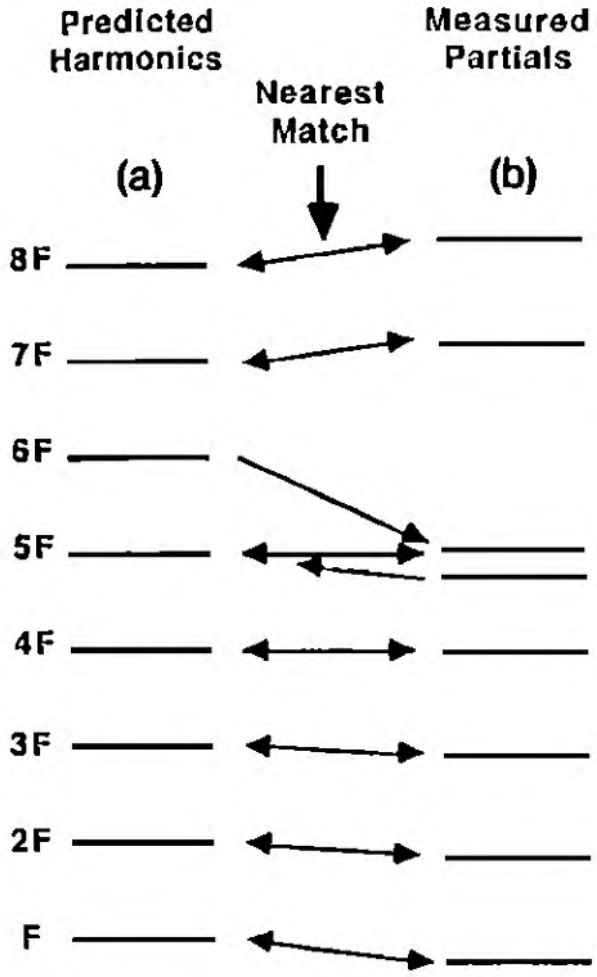


Рисунок 5.1 – обчислення похибки двосторонньої невідповідності

Алгоритм TWM, мінімум якого визначає значення $F_0$ для кожного семпла, базується на наступних трьох твердженнях:

- припущений гармонійний зв'язок між гармоніками вказує на те, що помилка невідповідності частоти в Гц між виміряними та передбаченими гармоніками повинні бути нормовані за частотою. Невідповідність у 10 Гц для компонентів поблизу 50 Гц гірша (20%), ніж невідповідність у 10 Гц для компонентів поблизу 5 кГц (0,2%). (b)
- Оскільки алгоритм ДПФ повертає інформацію із приблизно лінійною роздільною здатністю, дробова роздільна здатність покращується із збільшенням частоти. Таким чином, вищі гармоніки мають кращу дробову роздільну здатність, що може допомогти покращити оцінку відповідної основної частоти.
- Гучніші гармоніки, як правило, мають вище відношення сигнала до шуму, ніж тихіші гармоніки. Отже, вважається, що гучніші гармоніки мають більш надійні оцінки частоти, і їх присутність менш імовірна через шум або інші помилкові явища.

Виходячи з наведених вище тверджень, була постульована функція зважування помилок $E_\omega$. Для виміряної гармоніки $n$, $E_\omega$ можна виразити як функцію різниці частот між її частотою та частотою найближчої передбачуваної гармоніки ($\Delta f_n$), виміряної амплітуди та частоти гармоніки ($a_n$ і $f_n$,) та максимальної амплітуди будь-якої гармоніки у цьому семплі (*maximumMagnitude*). Крім того, із вищезазначених міркувань ясно, що функція помилки $E_\omega$ повинна мати наступні властивості:

- Максимальна похибка виникає, якщо значення $\Delta f_n$ / $f_n$ велике або якщо значення $\Delta f_n$ / $f_n$ мале та значення $a_n$ / *maximumMagnitude* невелике.
- Мінімальна похибка виникає, коли значення $\Delta f_n$ / $f_n$ мале та значення $a_n$ / *maximumMagnitude* велике.

Процедура обчислення TWM для кожного семпла може бути поданою у наступному вигляді:

1. Отримати *K* виміряних гармонік із амплітудами $A_k$ та відповідними частотами $f_k$ із певного семпла ДПФ; визначити *maximumMagnitude* як *maximumMagnitude* =*max* $\{A_k\}$ та $f_{max}$ як $f_{max}$ =*max* $\{f_k\}$
2. Вибрати $f_{fund}$, пробну основну частоту (спочатку нижче відомого діапазону частот вхідного сигнала) та обчислити частоти *inputSize* гармонік як $f_n = nf_{\text{fund}}$, де *inputSize* = *ceil* $\{f_{max} / f_{fund}\}$ – найменш ціле число, більше за $f_{max} / f_{fund.}$
3. Для кожної прогнозованої $f_n$ визначити відповідну частоту $f_k$, яка є найближчою до першої. Іншими словами, для кожної $f_n$ виберати $f_k$ таку, щоб мінімізувати $\Delta f_n = |f_n - f_k|$. Для *k*, що відповідає найближчій частоті, встановити $a_n = A_k$.
4. Обчислити похибку передбачуваної до виміряної невідповідності згідно з формулою зважування:

$$\begin{aligned}\mathrm{Err}_{p\to m} &= \sum_{n=1}^{N} E_w(\Delta f_n, f_n, a_n, A_{\max}) \\ &= \sum_{n=1}^{N} \Delta f_n \cdot (f_n)^{-p} + \left(\frac{a_n}{A_{\max}}\right) \\ &\quad \times [q\Delta f_n \cdot (f_n)^{-p} - r].\end{aligned} \tag{5.1}$$

Емпіричним шляхом було встановлено значення параметрів *frequencyWeight*,*magnitudeWeight* та: *frequencyWeight* = 0.5, *magnitudeWeight* = 1.4, *rmagnitudeScale* = 0.5.

5. Для кожної з $f_k$ визначити відповідну передбачену частоту $f_n$, яка є найближчою до неї. Іншими словами, для кожної $f_k$ вибрати $f_n$, щоб мінімізувати $\Delta f_k = |f_n - f_k|$. Для *n*, що відповідає найближчій частоті, встановити $a_k = A_n$.

6. Обчислити похибку виміряної до передбачуваної невідповідності згідно з формулою зважування:

$$\begin{aligned}\mathrm{Err}_{m\to p} &= \sum_{k=1}^{K} E_w(\Delta f_k, f_k, a_k, A_{\max}) \\ &= \sum_{k=1}^{K} \Delta f_k \cdot (f_k)^{-p} + \left(\frac{a_k}{A_{\max}}\right) \\ &\quad \times [q\Delta f_k \cdot (f_k)^{-p} - r],\end{aligned} \tag{5.2}$$

Значення *frequencyWeight*, *magnitudeWeight* та *rmagnitudeScale* також становлять 0.5, 1.4 та 0.5 відповідно. Тоді загальна похибка TWM для передбачуваної $f_{fund}$

$$\mathrm{Err}_{\mathrm{total}} = \mathrm{Err}_{p\to m}/(N) + \rho\mathrm{Err}_{m\to p}/(K) \tag{5.3}$$

визначається за формулою

Емпіричним шляхом встановлено, що найкраще значення ρ = 0.33.

Кроки 2 – 6 повторюються для послідовності випробувальних фундаментальних частот, що покривають увесь діапазон вхідного сигнала. Приклад залежності $Err_{total}$ від $f_{fund}$ зображения на рисунку 5.2.

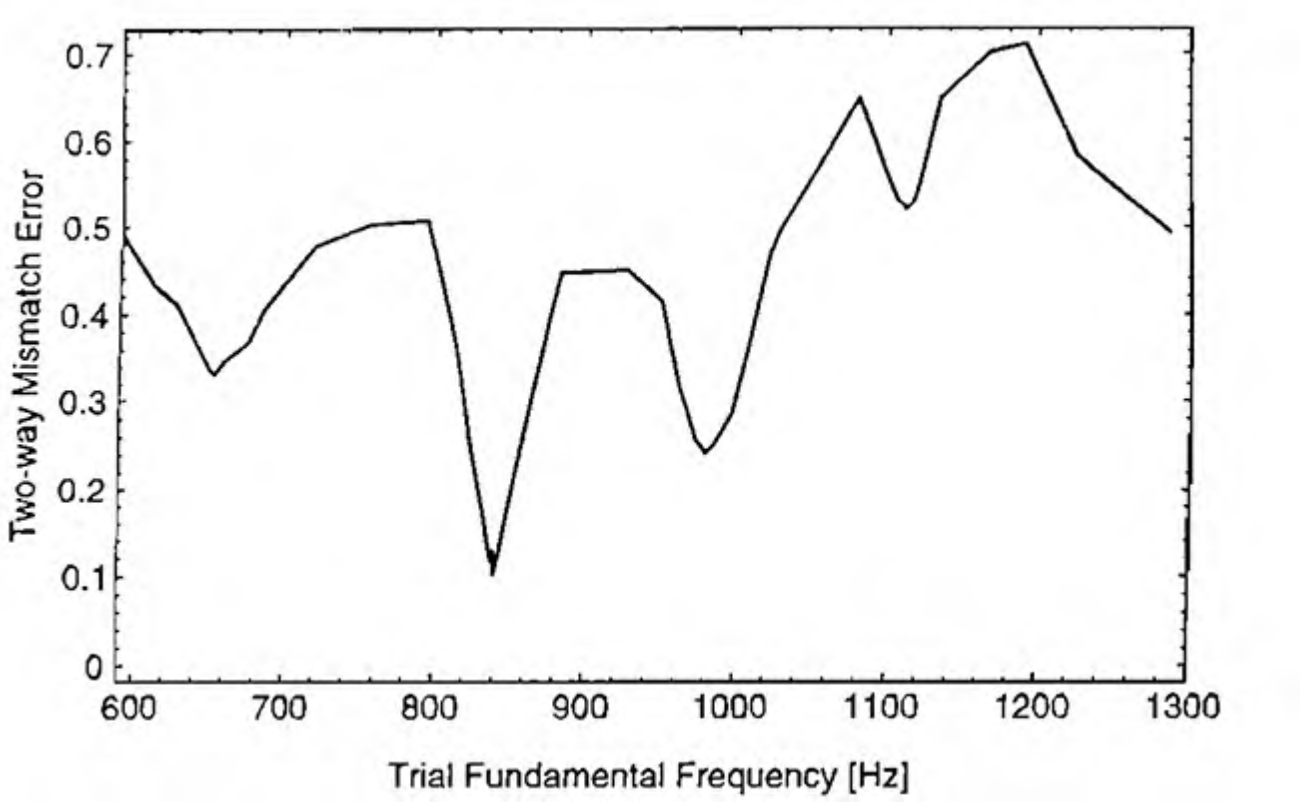


Рисунок 5.2 – залежність $Err_{total}$ від $f_{fund}$

У даному прикладі на рисунку 6.5 наявні декілька локальних мінімумів. Оцінкою $F_0$ кожного семпла вважається випробувальна основна частота із найменшою похибкою.

Процедура обчислення помилок TWM знаходить мінімальну та максимальну частоти для $f_{fund}$ та потім виконує глобальний пошук для мінімальної похибки невідповідності протягом усього діапазону частот. Цей процес повторюється, щоб випробувати частоту в рівномірних кроках (півтонах, тобто за коефіцієнтами 1.05946), починаючи з найнижчої частоти. Однак, така сітка часто недостатньо точна, щоб виявити справжню мінімальну помилку TWM, тому після цього досліджуються околи кожного локального мінімуму зі спадним кроком,

поки $Err_{total}$ не стає нижчою за встановлене значення. У кінці обирається оцінка $F_0$, що відповідає найменшому з локальних мінімумів.

Процедура TWM має три параметри, що можуть бути встановленими користувачем, щоб налаштувати процедуру для характеристик вхідного сигнала.

- Діапазон пошуку $F_0$: мінімальна та максимальна частоти розширюють діапазон фундаментальних частот. Вибір широкого діапазону збільшує час виконання процедури та ймовірність виникнення помилок.
- Кількість передбачуваних гармонік *inputSize*: вибір великої кількості гармонік, підходить для звукових файлів із мінімальними фоновим шумом і реверберацією. Вибір малої кількості гармонік найкраще працює з шумними сигналами, обмежуючи вплив небажаних звуків у вимірюваному сигналі. Недоліком у цьому випадку є нижча роздільна здатність. Вибір *inputSize* також залежить від відомих або припущених спектральних характеристик вхідного сигнала.
- Показник *frequencyWeight* у функції помилки: параметр *frequencyWeight* регулює залежне від частоти зважування різниці частот. Значення *frequencyWeight* = 0.5 чудово підходить для багатьох випадків. Значення *frequencyWeight* = 1.0 показує кращі результати із сигналами, що мають багато реверберації, завдяки зменшенню акцентів на високочастотних компонентах із малою амплітудою.

У якості прикладу розглянемо простий виконання процедури TWM, взявши у якості вимірюваного сигнала послідовність частот: 200, 300, 500, 600, 700, 800 Гц. У цьому випадку визначимо, яка з частот є найкращою у якості фундаментальної: 50, 100 або 200 Гц. Припустимо також, що всі виміряні гармоніки мають однакові амплітуди. Використовуючи 50 Гц у формулах TWM, отримуємо $Err_{p\to m}$ = 122.58, $Err_{m\to frequencyWeight}$ = -3.0, $Err_{total}$ = 7.49. Вибір 100 Гц у якості $F_0$ дає $Err_{p\to m}$ = 32.0, $Err_{m\to frequencyWeight}$ = -3.0, $Err_{total}$ = 3.83. Вибравши 200 Гц у якості $F_0$, отримуємо $Err_{p\to m}$ = 10.0, $Err_{m\to frequencyWeight}$ = 30.66, $Err_{total}$ = 4.2. Як бачимо, мінімальна похибка спостерігається при *fFrequencies* = 100 Гц, тому ця частота є фундаментальною. Варто звернути увагу на те, що ані передбачувані до вимірюваних, ані вимірювані до передбачуваних помилок самі по собі не можуть однозначного визначити $F_0$. У розглянутому прикладі різниця між

мінімумами для 100 та 200 Гц не є великою, але якщо ми включимо гармоніку із частотою 100 Гц у вимірюваний спектр, то вимірювання значно покращаться. У цьому випадку похибка для 200 Гц приблизно однакова (4,0), тоді як похибка для 100 Гц зменшується до < 1.0, тобто відмінність результату залежить від ступеня, до якого $F_0$ можна інтерпретувати однозначно з початкового спектра[10].

# РОЗДІЛ 6. СТОХАСТИЧНА МОДЕЛЬ

Розглянемо людський спів. Будь-яке слово, яке вимовляє вокаліст під час виступу, є комбінацією голосних та приголосних звуків. За формування голосних звуків відповідають голосові зв'язки, тому такі звуки можна описати періодичними функціями та, як наслідок, застосувавши ДПФ, отримати спектр гармонік. А от приголосні звуки утворюються завдяки губам та різкому видиханню повітря. Такі звуки можна порівняти з ударом у барабан: вони не мають періодичних гармонік і їхні хвилі схожі на різкі короткі імпульси. Приблизний вигляд ударів у барабан зображено на рисунку 6.1.

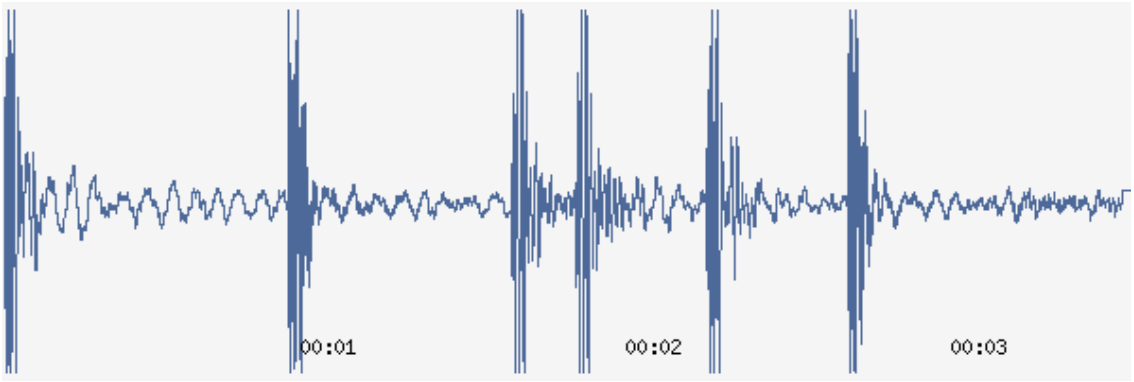


Рисунок 6.1 – форма хвилі удару в барабан

Описані раніше математичні процедури застосовуються до періодичних сигналів). Але якщо ми повернемося до людського співу, то зробимо наступний висновок: через те, що слова складаються із голосних і приголосних звуків, запис людського співу має одночасно і гармонійну, і негармонійну складові. Гармонійна модель утворюється завдяки обчисленню ДПФ, а негармонійна – шляхом знаходження стохастичної моделі. Стохастична модель будується за формулою

$$yst[n] = \sum_{k=0}^{N-1} u[n]h[n-k] \qquad (6.1).$$

Тут *yst*[*n*] – вихідна модель, *u*[*n*] – «білий» шум (сигнал, що складається з випадкових звуків на діапазоні частот 1 – 20000 Гц), *h*[*n*-*k*] – імпульсна характеристика фільтра, що апроксимує вхідний сигнал[11]. Візуалізована стохастична модель зображена на рисунку 6.2.

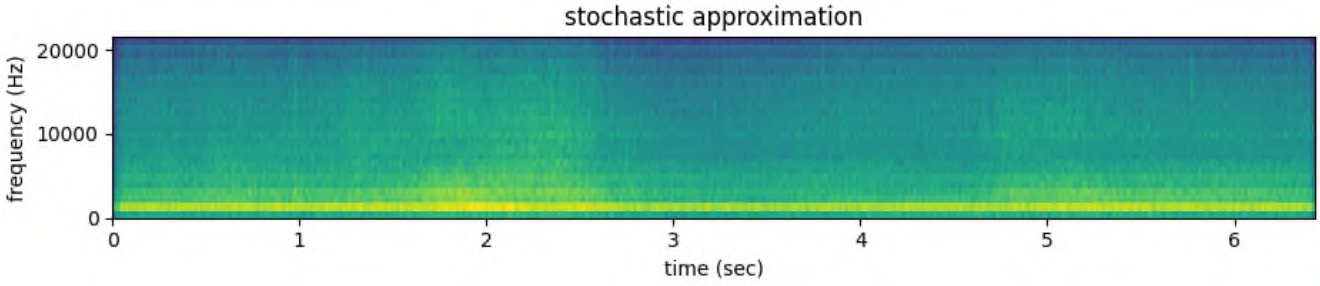


Рисунок 6.2 – візуалізація стохастичної моделі

# РОЗДІЛ 7. УТВОРЕННЯ ВИХІДНОГО СИГНАЛА В ІНШІЙ ТОНАЛЬНОСТІ

Як ми знаємо, дискретне перетворення Фур'є перетворює скінченну послідовність рівних за довжиною семплів функції у послідовність такої ж довжини з рівновіддаленими семплами частотного спектра з дискретним часом, яка є комплексною функцією частоти. Інтервал, на якому обчислюється ДПФ, є зворотним значенням тривалості вхідної послідовності. Зворотне перетворення Фур’є – це ряд Фур'є, що використовує семпли ДПФ як коефіцієнти складних синусоїд на відповідних частотах ДПФ. Він має ті самі значення, що і вхідний сигнал. Тому ДПФ називають поданням у частотній області вхідної послідовності. Якщо вхідна послідовність охоплює всі ненульові значення функції, її ДПФ є безперервним і періодичним. Якщо вхідна послідовність являє собою один цикл періодичної функції, ДПФ надає всі ненульові значення одного циклу.

Отримана за допомогою операцій, описаних у попередніх розділах, гармонійно-стохастична модель може бути конвертованою у складну синусоїдальну функцію, що є ідентичною до вхідного сигнала. Це можливо завдяки тому, що кожний семпл вхідного сигнала містить ряд гармонік, який відображується у спектрі. Тому ми можемо об’єднати всі гармоніки із спектра кожного семпла та отримати один семпл вихідного сигнала. Повторивши цей процес для кожного семпла спектра, отримуємо вихідний сигнал, ідентичний вхідному. Така послідовність дій називається зворотним дискретним перетворенням Фур’є й описується формулою[6]:

$$x[n] = \sum_{k=0}^{N-1} X[k] e^{\frac{2\pi i}{N} kn}, n \in \mathbb{Z} \qquad (7.1)$$

Завдяки тому, що гармонійно-стохастична модель містить у собі інформацію про всі гармонійні складові вхідного сигнала, можна змістити весь спектр частот, отримавши новий вихідний сигнал в іншій тональності. Оскільки спектр має інформацію про частоти всіх гармонік, за допомогою звичайного множення можна скорегувати висоту будь-якої ноти. Наразі розроблена система підтримує можливість змінити тональність усього сигнала, а не певної ноти.

У музиці існує поняття «інтервал», що позначає відстань від однієї ноти до іншої. Наприклад, відношення октави до тоніки позначається як 2:1, відношення мінорної терції до тоніки – як 1.1892:1 тощо (рис. 7.1). Тобто знаючи різницю між нотами вхідного сигнала і бажаними нотами, можна перенести вхідний сигнал в іншу тональність.

| Interval | Equal Temperament Frequency Ratio | Harmonic Series Frequency Ratio |
|---|---|---|
| Unison | ≃ 1.0000 | 1.0000 ≃ 1/1 |
| Minor Second | ≃ 1.0595 | 1.0909 ≃ 12/11 |
| Major Second | ≃ 1.1225 | 1.1250 ≃ 9/8 |
| Minor Third | ≃ 1.1892 | 1.2000 ≃ 6/5 |
| Major Third | ≃ 1.2599 | 1.2500 ≃ 5/4 |
| Perfect Fourth | ≃ 1.3348 | 1.3333 ≃ 4/3 |
| Tritone | ≃ 1.4142 | 1.4000 ≃ 7/5 |
| Perfect Fifth | ≃ 1.4983 | 1.5000 ≃ 3/2 |
| Minor Sixth | ≃ 1.5874 | 1.6000 ≃ 8/5 |
| Major Sixth | ≃ 1.6818 | 1.6667 ≃ 5/3 |
| Minor Seventh | ≃ 1.7818 | 1.7500 ≃ 7/4 |
| Major Seventh | ≃ 1.8897 | 1.8333 ≃ 11/6 |
| Octave | ≃ 2.0000 | 2.0000 ≃ 2/1 |

Рисунок 7.1 – співвідношення музикальних інтервалів

# РОЗДІЛ 8. ІНСТРУКЦІЯ КОРИСТУВАЧА

Не дивлячись на те, що інтерфейс програми доволі простий, він дає можливість робити потужні перетворення з аудіо-сигналами. Програма запускається у вигляді, представленому на рисунку 8.1.

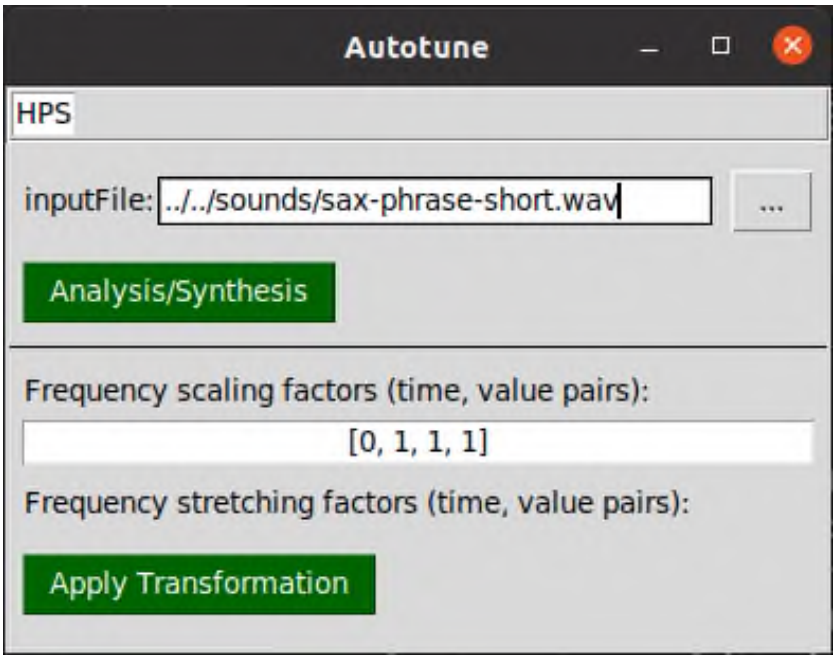


Рисунок 8.1 – стартовий екран програми

Кнопка, виділена на рисунку 8.2, дає можливість обрати аудіо-файл для виконання перетворення.

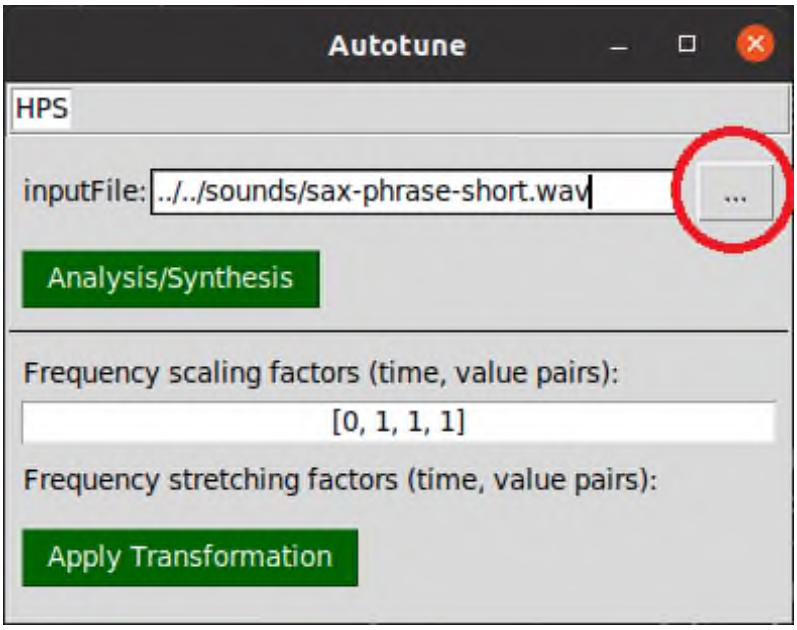


Рисунок 8.2 – вибір файла

Натискання на кнопку “Analysis/Synthesis” побудує гармонічно-стохастичну модель та конвертує її у вихідний файл для порівняння із вхідним (рисунок 8.3).

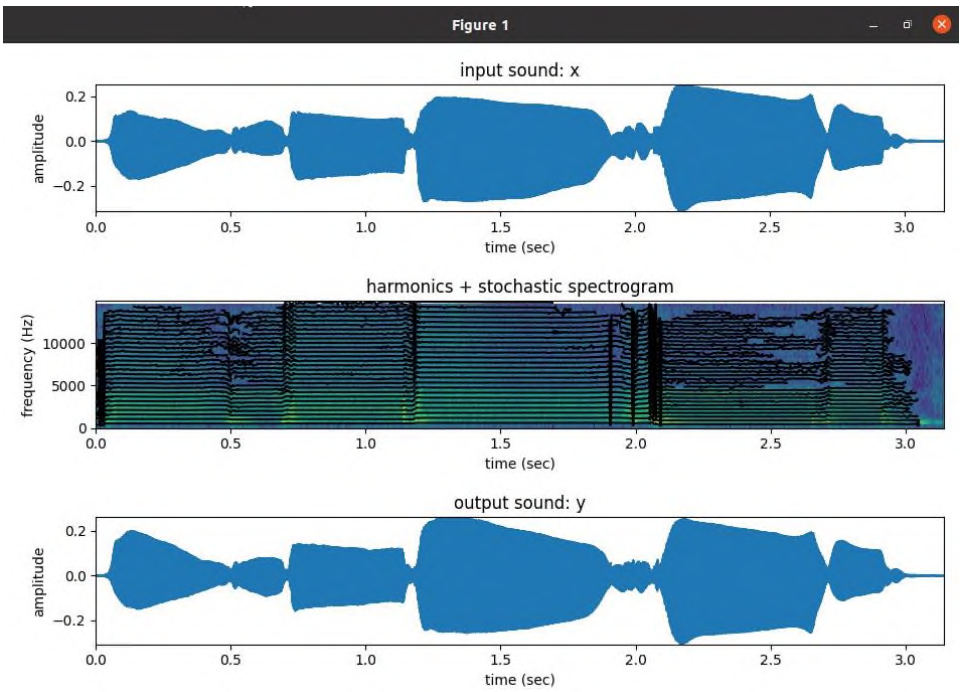


Рисунок 8.3 – побудова моделі

На рисунку 8.3 перший графік – функція, що описує вхідний сигнал, другий графік – спектр із виділеними на ньому гармоніками, третій графік – функція, що описує конвертовану в аудіо-сигнал гармонійно-стохастичну модель.

У виділеній на рисунку 8.4 області користувач може обрати, у скільки разів потрібно змінити висоту сигнала. Тут 0 та 1 – початок та кінець сигнала відповідно, 2 – константа, що позначає підвищення вхідного сигнала у два рази (на октаву).

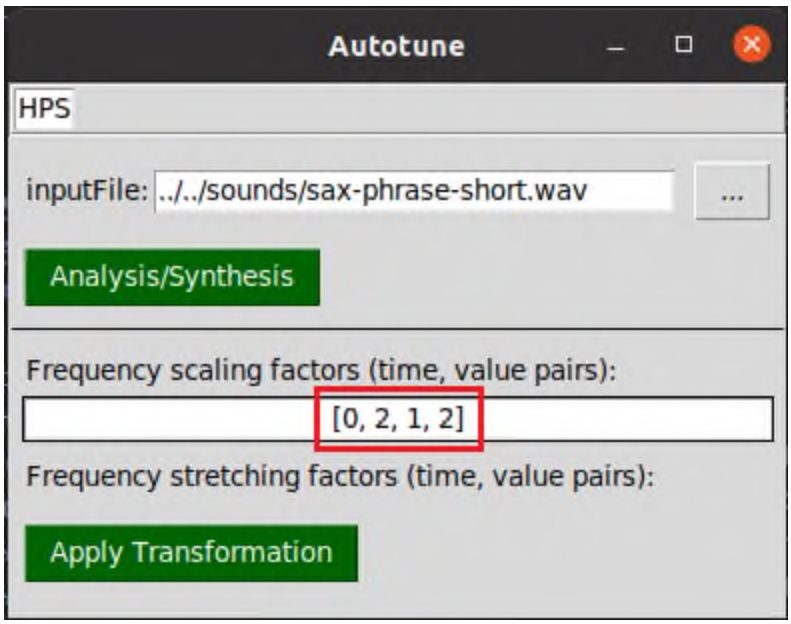


Рисунок 8.4 – вибір висоти вихідного сигнала

Натискання на клавішу “Apply Transformation” побудує гармонійно-стохастичну модуль у новій тональності (через октаву) та конвертує отриману модель у вихідний аудіо-сигнал. На рисунку 8.5 перший графік – візуалізація гармонійно-стохастичної моделі, другий – функція, що описує вихідний аудіо-сигнал.

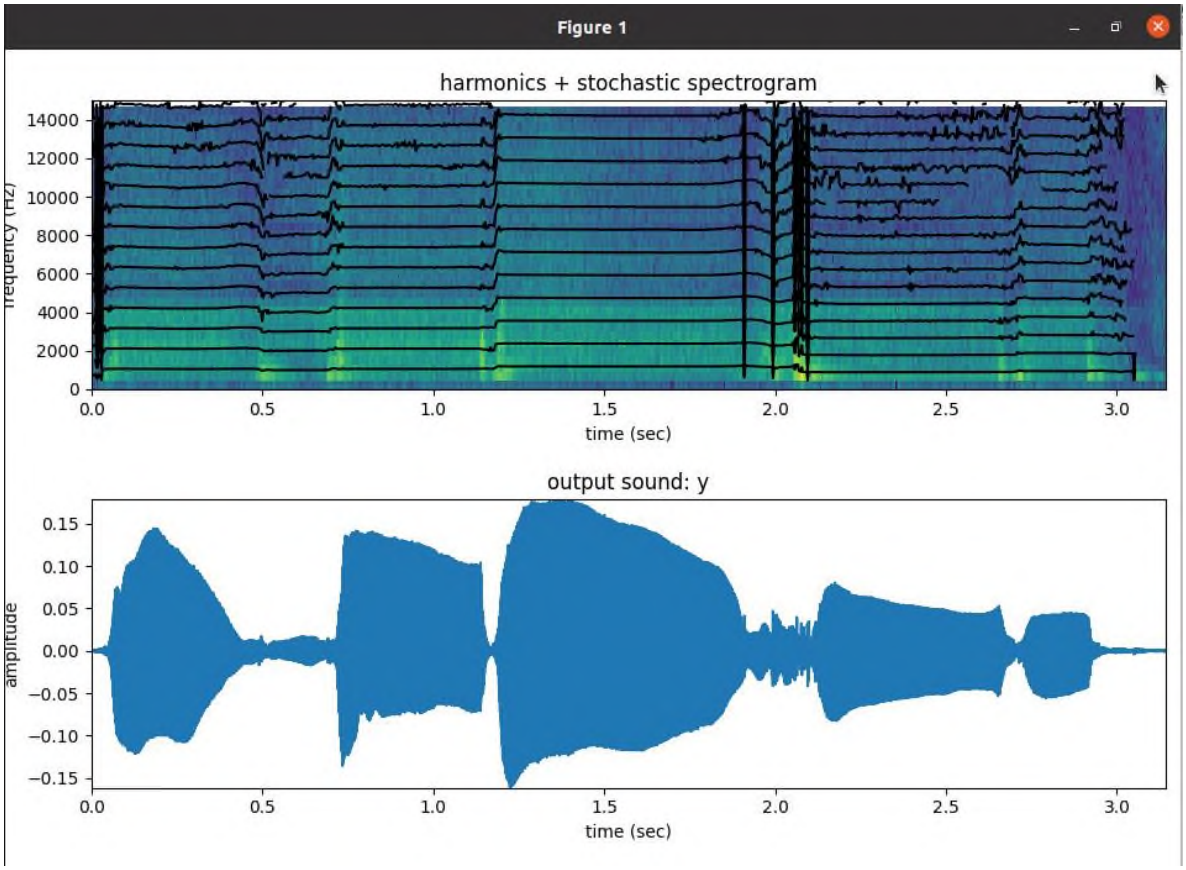


Рисунок 8.5 – гармонійно-стохастична модель вихідного сигнала

# РОЗДІЛ 9. ВИСНОВКИ

Під час роботи над розглянутою системою було проаналізовано сучасний стан предметної області, а також її математичний, фізичний та музичний аспекти. Були покращені навички використання комп’ютерних наук і технологій у практичному житті. Описані технології були використані для розробки програми, що виконує поставлену задачу: зміна висоти вхідного сигнала без втрати якості.

Представлена у роботі програма має потенціал для розвитку, адже її можна покращити, додавши можливості корегувати висоту окремих нот та застосовувати до вхідного сигнала інші звукові ефекти, які використовуються у музичній сфері.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Wikipedia. Periodic function [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Periodic_function.
2. Wikipedia. Signal [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Signal.
3. Wikipedia. Analog signal [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Analog_signal.
4. Wikipedia. Digital signal [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Digital_signal.
5. Wikipedia. Equalization (audio) [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Equalization_(audio).
6. Wikipedia. Discrete Fourier transform [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Discrete_Fourier_transform.
7. Wikipedia. Euler's formula [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Euler%27s_formula.
8. Wikipedia. Frequency domain [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Frequency_domain.
9. Wikipedia. Fast Fourier transform [Електронний ресурс] / Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Fast_Fourier_transform.
10. Maher R. Fundamental frequency estimation of musical signals using a two-way mismatch procedure / R. Maher, J. Beauchamp. – 1994.
11. Doets P. STOCHASTIC MODEL OF A ROBUST AUDIO FINGERPRINTING SYSTEM / P. Doets, R. Lagendijk.

# ДОДАТОК А
## Знаходження основної частоти

```
def fundamentalFrequencyDetection(x, fs, w, inputSize, hopSize, t,
minimumf0, maximumf0, f0et):
      spectrumSize = inputSize//2
      halfSizeRounding = int(math.floor((w.size+1)/2))
      halfSizeFloor = int(math.floor(w.size/2))
      x = np.append(np.zeros(halfSizeFloor),x
      x = np.append(x,np.zeros(halfSizeRounding
      currentSample = halfSizeRounding
      lastSample = x.size – halfSizeRounding
      w = w / sum(w)
      fFrequencies = []
      f0stable = 0
      while currentSample<lastSample:
            x1 = x[currentSample-
            halfSizeRounding:currentSample+halfSizeFloor]
            magnitude, phase = dft.dft_analysis(x1, w, inputSize)
            peakLocation = Functions.peakDetection(magnitude, t)
            interpPeakLocation, interpMagnitude =
            Functions.peakInterp(magnitude, phase, peakLocation)
            interpFrequency = fs * interpPeakLocation/inputSize
            fundamentalFrequency = Functions.twm_init(interpFrequency,
            interpMagnitude, f0et, minimumf0, maximumf0, f0stable)
            if ((f0stable==0)&(fundamentalFrequency>0)) \
            or ((f0stable>0)&(np.abs(f0stable-
            fundamentalFrequency)<f0stable/5.0)):
                  f0stable = fundamentalFrequency
            else:
                  f0stable = 0
            fFrequencies = np.append(fFrequencies, fundamentalFrequency)
            currentSample += hopSize
      return fFrequencies
```

# ДОДАТОК Б
## Алгоритм двосторонньої невідповідності

```
def twm_init(peakFrequency, peakMagnitude, maximumError, minf0, maxf0,
f0t=0):
      if (peakFrequency.size < 3) & (f0t == 0
            return 0

      fundamentalCandidate = np.argwhere((peakFrequency>minf0) & (
      peakFrequency<maxf0))[:,0]
      if (fundamentalCandidate.size == 0):
            return 0
      fundamentalCandidateFrequencies = peakFrequency[fundamentalCandidate]
      f0cm = peakMagnitude[fundamentalCandidate]

      if f0t>0:
            closePeaks = np.argwhere(np.abs(fundamentalCandidateFrequencies-
            f0t)<f0t/2.0)[:,0]
            maximumCandidate = np.argmax(f0cm)
            maximumCandidateFrequency =
            fundamentalCandidateFrequencies[maximumCandidate]%f0t
            if maximumCandidateFrequency > f0t/2:
                  maximumCandidateFrequency = f0t -
                  maximumCandidateFrequency
            if (maximumCandidate not in closePeaks) and
            (maximumCandidateFrequency>(f0t/4)):
                  closePeaks = np.append(maximumCandidate, closePeaks)
            fundamentalCandidateFrequencies =
            fundamentalCandidateFrequencies[closePeaks]

      if (fundamentalCandidateFrequencies.size == 0):
            return 0

      fundamentalFrequency, fundamentalFrequencyError = twm(peakFrequency,
      peakMagnitude, fundamentalCandidateFrequencies)
      if (fundamentalFrequency>0) and
      (fundamentalFrequencyError<maximumError):
            return fundamentalFrequency
      else:
            return 0

def twm(peakFrequency, peakMagnitude, fundamentalCandidateFrequencies):
      frequencyWeight = 0.5
      magnitudeWeight = 1.4
      rmagnitudeScale = 0.5
      errorWeight = 0.33
      maximumMagnitude = max(peakMagnitudes)
      maxPeaksNumber = 10
      harmonic = np.matrix(fundamentalCandidateFrequencies)
      errors = np.zeros(harmonic.size)
      measuredToPredicted = min(maxPeaksNumber, peakFrequencies.size)
      for i in range(0, measuredToPredicted):
            differences = harmonic.T * np.ones(peakFrequencies.size)
            differences = abs(differences - np.ones((harmonic.size,
            1))*peakFrequencies)
            distance = np.amin(differences, axis=1)
```

```
        peakLocations = np.argmin(differences, axis=1)
        difpond = np.array(distance) * (np.array(harmonic.T)**(-f
        requencyWeight))
        peakMagnitude = peakMagnitudes[peakLocations]
        magnitudeCoefficient = 10**((peakMagnitude-maximumMagnitude)/20)
        errors = errors + (difpond +
        magnitudeCoefficient*(magnitudeWeight*difpond-
        rmagnitudeScale)).T
        harmonic = harmonic+fundamentalCandidateFrequencies

errorMeasuredToPredicted = np.zeros(harmonic.size)
measuredToPredicted = min(maxPeaksNumber, peakFrequencies.size)

for i in range(0, fundamentalCandidateFrequencies.size):
        nharm =
        np.round(peakFrequencies[:measuredToPredicted]/fundamentalCandid
        ateFrequencies[i])
        nharm = (nharm>=1)*nharm + (nharm<1)
        distance = abs(peakFrequencies[:measuredToPredicted] -
        nharm*fundamentalCandidateFrequencies[i])
        difpond = distance * (peakFrequencies[:measuredToPredicted]**(-
        frequencyWeight))

        peakMagnitude = peakMagnitudes[:measuredToPredicted]
        magnitudeCoefficient = 10**((peakMagnitude-maximumMagnitude)/20)
        errorMeasuredToPredicted[i] = sum(magnitudeCoefficient *
        (difpond + magnitudeCoefficient*(magnitudeWeight*difpond-
        rmagnitudeScale)))

totalError = (errors[0]/measuredToPredicted) +
(errorWeight*errorMeasuredToPredicted/measuredToPredicted)
f0index = np.argmin(totalError)
f0 = fundamentalCandidateFrequencies[f0index]

return f0, totalError[f0index]
```

# ДОДАТОК В
## Зміна частоти вхідного сигнала

```
def hpsTimeScale(hfreq, hmag, stocEnv, coefficient):
      inputSize = hfreq[:,0].size
      maximumInputValue = max(coefficient[::2])
      maximumOutputValue = max(coefficient[1::2])
      outputSize = int(inputSize*maximumOutputValue/maximumInputValue)
      inFrames = (inputSize-1)*coefficient[::2]/maximumInputValue
      outputTimeValues = outputSize*coefficient[1::2]/maximumOutputValue
      scaleEnvelope = interp1d(outputTimeValues, inFrames, fill_value=0)
      samplesIndexes = scaleEnvelope(np.arange(outputSize))
      outpuStartFrequency = hfreq[int(round(samplesIndexes[0])),:]
      outpuStartMagnitude = hmag[int(round(samplesIndexes[0])),:]
      outpuStartStochasticEnvelope =
      stocEnv[int(round(samplesIndexes[0])),:]
      for l in samplesIndexes[1:]:
            outpuStartFrequency = np.vstack((outpuStartFrequency,
            hfreq[int(round(l)),:]))
            outpuStartMagnitude = np.vstack((outpuStartMagnitude,
            hmag[int(round(l)),:]))
            outpuStartStochasticEnvelope =
            np.vstack((outpuStartStochasticEnvelope,
            stocEnv[int(round(l)),:]))

      return outpuStartFrequency, outpuStartMagnitude,
      outpuStartStochasticEnvelope
```

# ДОДАТОК Г
## Генерація вихідного файла

```
def inverse_dft(magnitude, phase, windowSize):
      spectrumSize = magnitude.size
      inputSize = (spectrumSize-1)*2

      halfSizeByRounding = int(math.floor((windowSize+1)/2))
      halfSizeByFloor = int(math.floor(windowSize/2))

      output = np.zeros(windowSize)
      outputSpectrum = np.zeros(inputSize, dtype = complex)
      outputSpectrum[:spectrumSize] = 10**(magnitude/20) * np.exp(1j*phase)
      outputSpectrum[spectrumSize:] = 10**(magnitude[-2:0:-1]/20) *
      np.exp(-1j*phase[-2:0:-1])

      buffer = np.real(ifft(outputSpectrum))
      output[:halfSizeByFloor] = buffer [-halfSizeByFloor:]
      output[halfSizeByFloor:] = buffer [:halfSizeByRounding]

      return output
```