Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики
Факультету інформатики

Курсова робота на тему:

“Розпізнавання акордів”

Текстова частина до курсової роботи
за спеціальністю „Системний аналіз” 124

Керівник курсової роботи
к. фіз. - мат. наук., доц. Чорней Р. К.
_____
(підпис)
“____” _____ 2020 р.
Виконав студент 5 курсу
Андрущак Г. С.
“____” _____ 2020 р.

Київ 2020

Тема: Розпізнавання акордів.

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи.	10.10.2019	
2.	Пошук та збір тематичних матеріалів.	20.11.2019	
3.	Ознайомлення з відповідними матеріалами та обдумування структури роботи.	25.12.2019	
4.	Написання вступу та плану роботи.	31.12.2019	
5.	Вивчення відповідних матеріалів, що необхідні для написання першого розділу роботи, та безпосереднє його написання.	03.02.2020	
5.	Опрацювання необхідних матеріалів та написання другого розділу роботи.	29.02.2020	
6.	Вивчення потрібних джерел та завершення роботи над третім розділом роботи.	24.03.2020	
7.	Коректне оформлення роботи відповідно до вимог написання курсової роботи.	29.03.2020	
8.	Створення презентації та написання доповіді для захисту роботи.	5.04.2020	
9.	Подання та аналіз попередньої версії роботи з керівником.	7.04.2020	
10.	Корегування роботи згідно із зауваженнями керівника.	15.04.2020	
11.	Захист курсової роботи.	18.05.2020	

Студент Андрущак Г. С.

Керівник к. фіз. - мат. наук., доц. Чорней Р. К.

“_____” _____

Зміст

Вступ 4

РОЗДІЛ 1. Загальна інформація про дослідження та обробку музики. 5

РОЗДІЛ 2. Теорія музики та її представлення. 7

РОЗДІЛ 3. Розпізнавання нот та акордів. 12
3.1. Розпізнавання нот. 12
3.2. Розпізнавання акордів 13
3.2.1. Розпізнавання акордів на базі шаблонів. 13
3.2.2. Розпізнавання акордів на базі прихованої марковської моделі 14

РОЗДІЛ 4. Розпізнавання акордів на базі шаблонів. 15

Висновки 18
Література 19
Додатки 20

Вступ

Актуальність теми. На сьогоднішній день, дослідження та обробка музики є молодою, проте пріоритетною дослідницькою галуззю, оскільки алгоритми обробки, дослідження, розпізнавання та пошуку музики ми використовуємо чи не щодня.

Методи дослідження. Перетворення Фур'є, дискретне перетворення Фур'є, алгоритм швидкого перетворення Фур'є на базі мови програмування Python 3, алгоритм розпізнавання акордів на базі шаблонів та його імплементація на базі мови програмування Python 3.

Практичне значення роботи. Програма дозволяє користувачеві визначити з яких акордів складається аудіофайл.

Робота складається зі вступу, чотирьох розділів, висновку, списку літератури та додатків. У першому розділі описується загальна інформація про дослідження та обробку музики. У другому розділі роботи досліджується теорія музики та її представлення. Третій розділ присвячено розпізнаванню нот та акордів. У четвертому розділі описується розпізнавання акордів на базі шаблонів. Загальний обсяг роботи становить 34 сторінки. Робота містить 5 рисунків та 3 додатки. Список використаної літератури налічує 3 найменування.

РОЗДІЛ 1. Загальна інформація про дослідження та обробку музики

Музика є всюдисущою і життєво важливою частиною життя мільярдів людей у всьому світі. Музичні твори та виступи є одними з найскладніших наших культурних артефактів, і емоційна сила музики може торкнутися нас дивовижними та глибокими способами. Музика охоплює величезний спектр форм і стилів: від простих, не супроводжуваних народних пісень, до популярної та джазової музики, до симфоній для повних оркестрів. Цифрова революція в розповсюдженні та зберіганні музики одночасно викликала величезний інтерес та увагу до способів застосування інформаційних технологій до такого типу контенту. Від перегляду особистих колекцій до відкриття нових виконавців, до управління та захисту прав творців музики, комп’ютери зараз глибоко залучені майже до кожного аспекту споживання музики, навіть не згадуючи їх життєво важливу роль у більшості сучасного музичного виробництва.

Незважаючи на важливість музики, обробка музики все ще є відносно молодою дисципліною порівняно з обробкою мови, дослідницькою сферою з давньою традицією. Власне, більш широке науково-дослідне співтовариство, представлене Міжнародним Товариством Пошуку Музичної Інформації (ISMIR), яке систематично займається широким спектром комп'ютерних тем музичного аналізу, обробки та пошуку, було сформовано у 2000 році. Традиційно, дослідження, засноване на музиці, здебільшого проводилось на основі символічних уявлень з використанням музичних позначень, або MIDI-репрезентацій. Через збільшення доступності оцифрованих аудіоматеріалів та вибуху обчислювальної потужності, автоматизована обробка звукових сигналів тепер все більше у фокусі дослідників.

Багато з цих досліджень спрямовані на розвиток технологій, які дозволяють користувачам отримувати доступ та досліджувати музику у різних її аспектах. Наприклад, методи аудіо дактилоскопії сьогодні інтегруються в

комерційні продукти, що допомагають користувачам організовувати свої приватні музичні колекції. Методи обробки музики використовуються в аудіоплеєрах, які підкреслюють поточні ступіні в межах нотного аркуша під час відтворення запису симфонії. На вимогу слухача автоматично подається додаткова інформація про мелодичну та гармонічну прогресії або ритм та темп. Інтерактивні музичні інтерфейси відображають структурні частини поточного музичного твору і дозволяють користувачам безпосередньо переходити до будь-якої ключової частини, наприклад, хорова секція, основна музична тема або сольна секція без нудного швидкого перемотування. Крім того, слухачі оснащені пошуковими системами, схожими на Google, які дозволяють їм досліджувати великі музичні колекції різними способами. Наприклад, користувач може створити запит, вказавши певну групу нот, або якийсь гармонійний, або ритмічний рисунок, насвистуючи мелодію, або настукуючи ритм, або просто обравши короткий уривок із запису. Система надає користувачеві список класифікованих музичних уривків із колекції, які музично пов'язані із запитом. В обробці музики однією з головних цілей є внести концепції, моделі, алгоритми, реалізації та оцінки для вирішення подібних проблем аналізу та пошуку [1].

За останні п’ятнадцять років обробка музики та пошук музичної інформації (MIP) перетворилися на яскраву та багатопрофільну область досліджень. Через різноманітність та багатство музики ця область об’єднує дослідників та студентів з багатьох областей, включаючи інформатику, аудіоінженерію, інформатику та музикознавство.

РОЗДІЛ 2. Теорія музики та її представлення

Музику можна представити різними способами та форматами. Наприклад, композитор може записати композицію у вигляді музичної партитури. У партитурі музичні символи використовуються для візуального кодування нот і того, як музиканти повинні грати на цих нотах. Друкована форма музичної партитури (рис. 1) також називається нотною формою. Оригінальним носієм цього представлення є папір, хоча вона також доступна на екранах комп'ютерів за допомогою цифрових зображень.

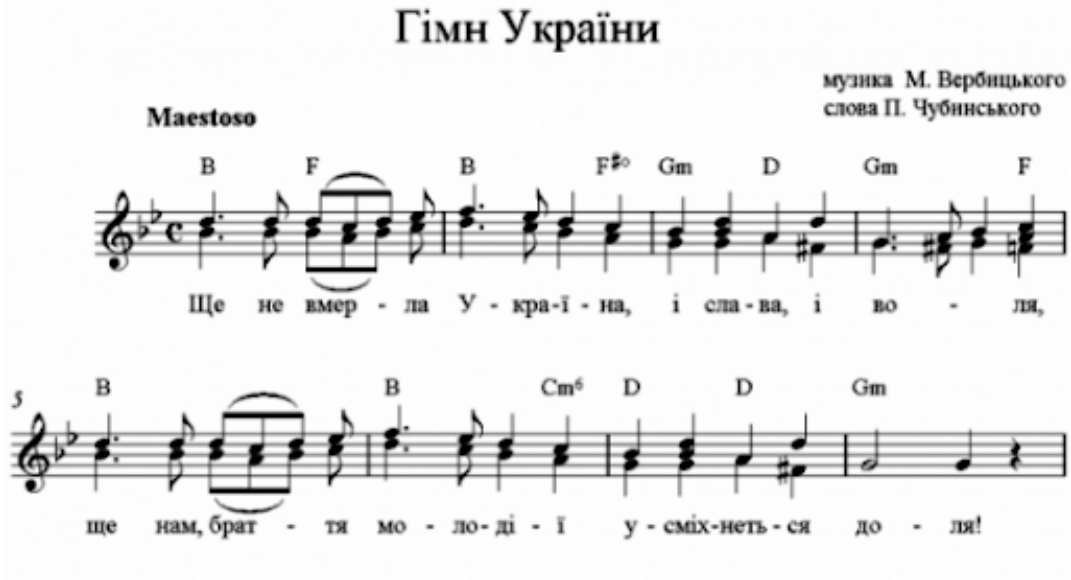


Рис. 1

Для електронних інструментів та комп'ютерів музика може передаватися за допомогою стандартних протоколів, таких як широко використовуваний протокол цифрового інтерфейсу музичного інструменту (MIDI) (рис. 2), де повідомлення про події задають тони, швидкості та інші параметри для генерування передбачених звуків. Термін **символічне представлення музики** використовується для позначення будь-якого машиночитаного формату даних, який явно представляє музичні сутності. Вони можуть варіюватися від нотних подій у певний час, як це стосується файлів MIDI, до графічних фігур із доданим музичним значенням, як це

стосується систем відображення музики. На відміну від символічних представлень, аудіопрезентації (рис. 3), такі як файли WAV або MP3, не вказують явно музичні події. Ці файли кодують акустичні хвилі, які генеруються, коли джерело (наприклад, інструмент) створює звук, який рухається до людського вуха як коливання тиску повітря.

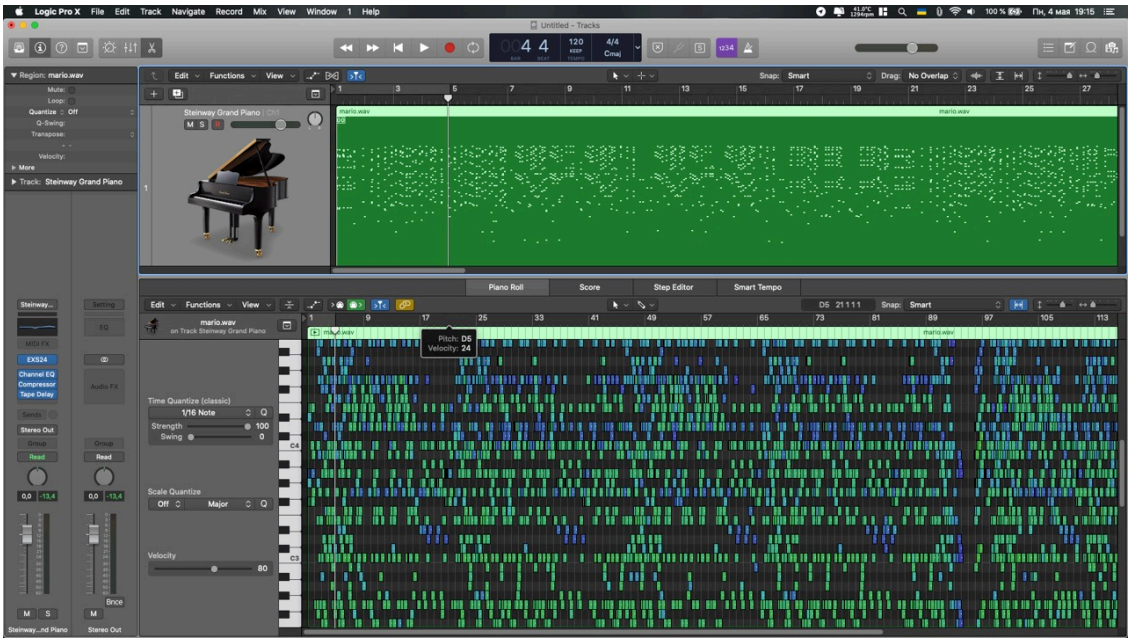


Рис. 2

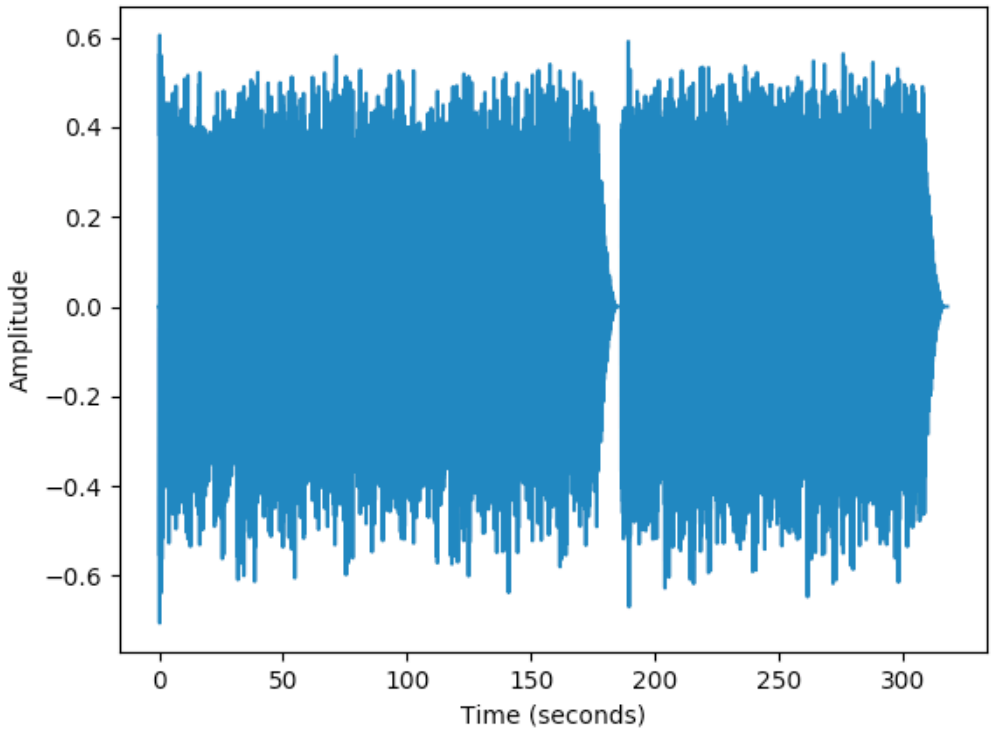


Рис. 3

Дві ноти з частотами у співвідношенні, що дорівнює будь-якій степені двійки, сприймаються як дуже схожі. Через це всі ноти з таким відношенням можуть бути згруповані в один клас. Це спостереження також призводить до фундаментального поняття октави, яке визначається як інтервал між однією музичною нотою та іншою з половиною або подвоюванням її основної частоти. Використовуючи це визначення, клас тонів - це множина усіх тонів або нот, які знаходяться на відстані цілого числа октав одна від одної [1].

Для того щоб описати музику за допомогою кінцевої кількості символів, потрібно дискретизувати простір усіх можливих тонів. Це призводить до поняття музичного строю, яке можна розглядати як скінченний набір репрезентативних тонів. Через близький октавний зв'язок тонів зазвичай вважається, що стрій охоплює одну октаву, а вищі або нижчі октави просто повторюють схему. Також музичний стрій можна задати діленням октавного простору на певну кількість ступенів строю. Елементи шкали часто просто називають нотами сторю і упорядковуються відповідно до відповідних тонів.

На сьогоднішній день найросповсюдженішим строєм є 12-ступеневий рівномірно темперований стрій (рис. 4 та рис. 5). Він є основним в європейській музиці з XIX століття. У цьому строї октава ділиться на певну кількість однакових ступенів, а саме 12. Вони віддалені один від одного на відстань хроматичного півтону $\left(1 : \sqrt[12]{2}\right)$. Нота A4 (*Ля* четвертої октави) відповідає фундаменальній частоті 440Гц, C0 (*До* нульової октави) – 16Гц.

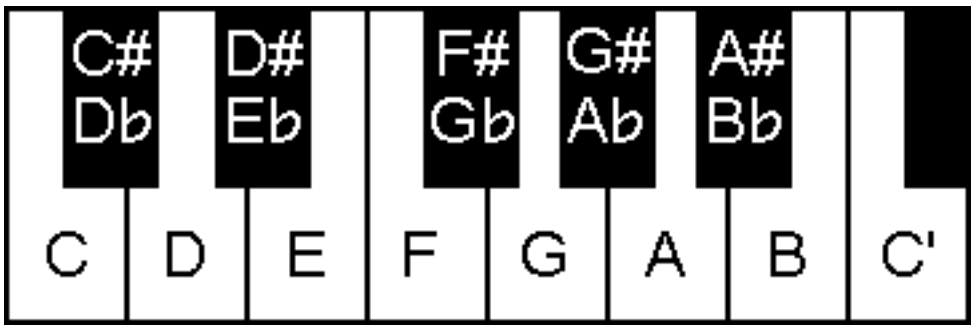


Рис. 4

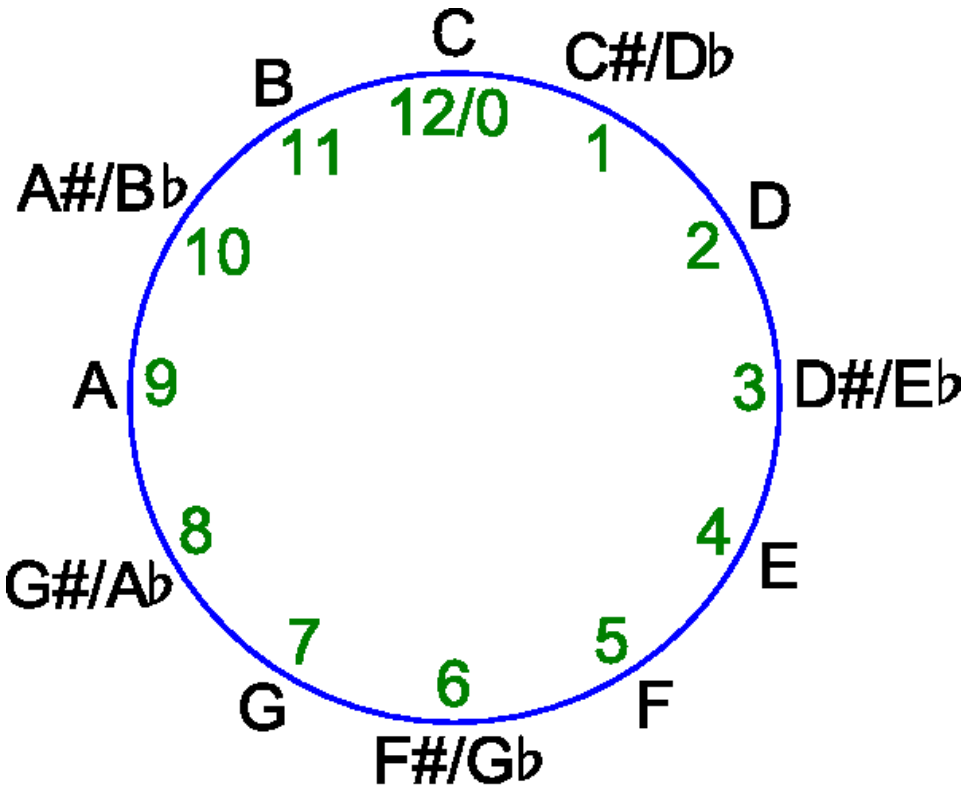
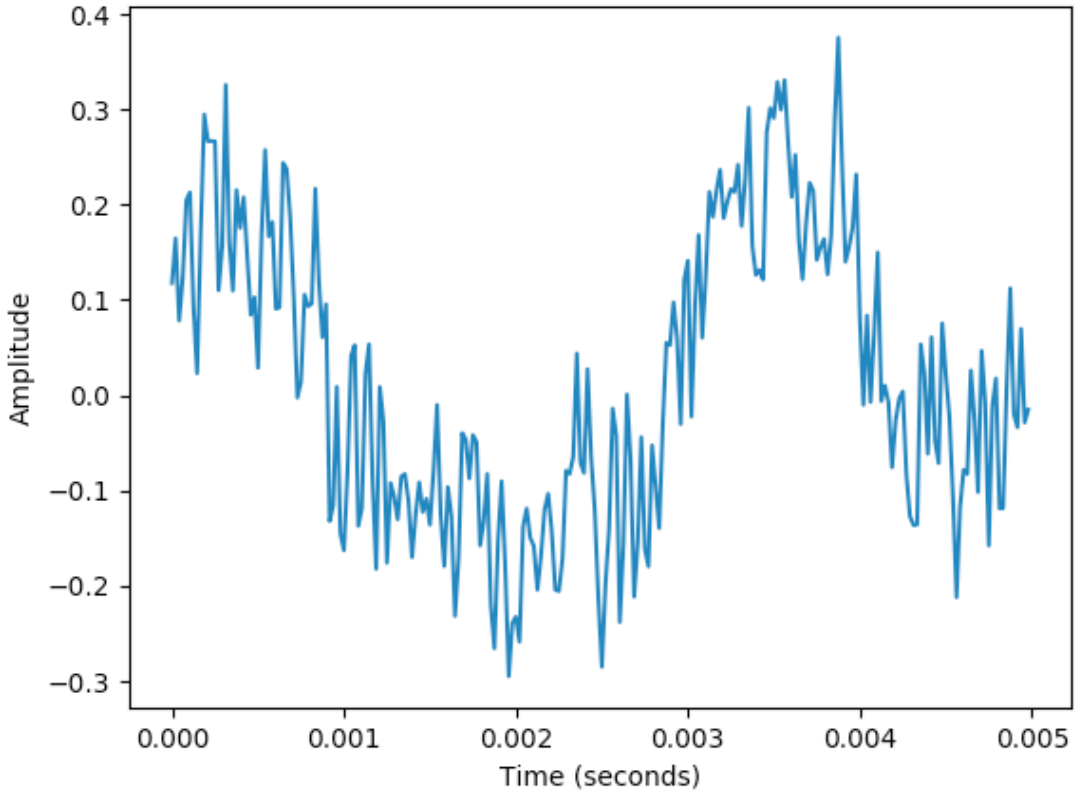


Рис. 5

Стандарт символічного представлення музики MIDI був розроблений у 1981 – 1983 рр. та досі активно використовується. Цей формат дає можливість електронним музичним інструментам взаємодіяти між собою чи комп'ютером та іншим MIDI-сумісним обладнанням, здійснювати з одного інструменту управління іншими. MIDI не передає та не генерує звук — натомість MIDI передає «повідомлення», такі як нота-вкл./нота-викл., висота (тон) та динаміка взятої ноти на інструменті; контрольні сигнали (CC) для таких параметрів як гучність, панорама, сигнали відліку часу для синхронізації темпу, тощо. Музичний інструмент приймає такі повідомлення і генерує звук. Інструментом може бути як реальний пристрій, наприклад, синтезатор, так і віртуальний - програма на комп'ютері.

Натомість, WAV (waveform audio format) зберігає значення амплітуд коливання з певною дискретизацією, наприклад 44100 значень на секунду, та є поширеним форматом для зберігання та відтворення аудіофайлів без стискання та втрати якості (lossless). Типовий зміст WAV файлу:


0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
Amplitude
0.000
0.001
0.002
0.003
0.004
0.005
Time (seconds)

РОЗДІЛ 3. Розпізнавання нот та акордів

3.1 Розпізнавання нот

У алгоритмах цифрової обробки сигналів широко застосовується перетворення Фур'є, а саме його дискретна версія. Перетворення Фур'є — інтегральне перетворення однієї комплекснозначної функції дійсної змінної на іншу. Перетворення Фур'є функції *f(t)* математично визначається як комплексна функція *F(ω)*, яка задається інтегралом

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

Обернене перетворення Фур'є задається виразом

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega$$

Дискретне перетворення Фур'є вимагає в якості входу дискретну функцію. Такі функції часто створюються шляхом дискретизації (вибірки значень з безперервних функцій). Дискретні перетворення Фур'є допомагають вирішувати диференціальні рівняння в частинних похідних і виконувати такі операції, як згортки.

Пряме перетворення:

$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-\frac{2\pi i}{N} kn} = \sum_{n=0}^{N-1} x_n \left[\cos\left(\frac{2\pi kn}{N}\right) - i \sin\left(\frac{2\pi kn}{N}\right) \right], (k = 0, \dots, N-1).$$

Обернене перетворення:

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} X_k \, e^{\frac{2\pi i}{N} kn} = \sum_{k=0}^{N-1} X_k \left[\cos\left(\frac{2\pi kn}{N}\right) + i \sin\left(\frac{2\pi kn}{N}\right) \right], (n = 0, \dots, N-1).$$

Позначення:

- N - кількість значень сигналу, виміряних за період, а також кількість компонент розкладання;
- $x_n,\ n = 0, \dots, N-1$ - виміряні значення сигналу, які є вхідними даними для прямого перетворення і вихідними для оберненого;
- $X_k,\ k = 0, \dots, N-1$ - N комплексних амплітуд синусоїдальних сигналів, що складають вихідний сигнал; є вихідними даними для прямого перетворення і вхідними для оберненого; оскільки амплітуди комплексні, то по ним можна обчислити одночасно і амплітуду, і фазу;

Знаючи частоту коливання тиску повітря, ми можемо точно сказати яка це нота. Приклад таких пар значень можна побачити у нашій програмі (Додаток А) [2].

3.2 Розпізнавання акордів

У музиці **гармонія** означає одночасне звучання різних нот, які утворюють згуртовану сутність у свідомості слухача. Основними складовими гармонії, принаймні у західній музичній традиції, є **акорди**, які є музичними конструкціями, які, як правило, складаються з трьох і більше нот. Наприклад, акорд C (*До* мажор) складається з нот C (*До*), E (*Мі*) та G (*Соль*).

Існують різні методи розпізнавання акордів, наприклад, на базі шаблонів, або на базі HMM (Прихована марковська модель).

3.2.1 Розпізнавання акордів на базі шаблонів

Типова система розпізнавання акордів складається з двох основних етапів. На першому етапі даний аудіозапис розрізається на кадри, і кожен кадр трансформується у нотний вектор. На другому кроці використовуються методи співставлення шаблонів, щоб порівняти кожен вектор з набором заздалегідь заданих моделей акордів. Найкраща міра подібності визначає акорд, що є у цьому кадрі [1].

3.2.2 Розпізнавання акордів на базі прихованої марковської моделі

Основна ідея - запровадити перехідну модель, яка виражає ймовірність переходу від одного акорда до іншого. Це призводить нас до концепції, відомої як прихована марковська модель (HMM). Поняття HMM широко застосовується в таких додатках, як розпізнавання мови, а також є фактично стандартним методом у більшості автоматизованих процедур розпізнавання акордів.

На основі ланцюга Маркова ми можемо обчислити ймовірність даного спостереження, що складається з послідовності станів або типів акордів. Однак у нашому сценарії розпізнавання акордів це не те, що нам потрібно. Замість того, щоб спостерігати послідовність типів акордів, ми спостерігаємо послідовність нотних векторів, які так чи інакше пов'язані з типами акордів. Іншими словами, послідовність станів не видно безпосередньо, а лише послідовність нечіткого спостереження, яка генерується на основі послідовностей станів. Крім того, замість обчислення ймовірності послідовності спостереження, мета - виявити взаємозв'язок між спостережуваними ознаками векторів та базовими типами акордів.

Далі ми розширимо поняття ланцюгів Маркова до статистичної моделі (HMM). Ідея полягає в тому, щоб представити зв'язок між спостережуваними характеристиками векторів та типами акордів (станами), використовуючи ймовірнісну структуру. Кожен стан оснащений функцією ймовірності, яка виражає ймовірність того, що певний тип акордів виводить певний вектор. В результаті ми отримуємо двошаровий процес, що складається з прихованого шару і шару, що спостерігається. Прихований шар створює послідовність станів, яку не можна спостерігати («прихована»), але генерує послідовність спостережень на основі функцій ймовірностей, залежних від стану [1].

Розділ 4. Розпізнавання акордів на базі шаблонів

Нашим шаблоном виступатиме матриця акордів, у якій кожному акорду відповідає 12-вимірний вектор-стовпчик, де **1**, якщо нота є частиною акорду, та **0**, якщо ні. В нашій програмі ми використовували таку матрицю:

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	Cm	C#m	Dm	D#m	Em	Fm	F#m	Gm	G#m	Am	A#m	Bm
B	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1
Bb	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0
A	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0
Ab	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0
G	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0
Gb	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1
F	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0
E	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0
Eb	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0
D	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1
Db	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0
C	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0

Використовуючи нашу імплементацію алгоритму швидкого перетворення Фур'є [2]

```
def fourier_transform(a):
    n = len(a)
    if n == 1:
        return a
    if n & (n - 1):
        raise ValueError('array length must be a power of 2')
    w_n = np.e**((2*np.pi*1j)/n)
    w = 1
    a_0 = []
    a_1 = []
    for i in range(n):
        if i % 2 == 0:
            a_0.append(a[i])
        else:
            a_1.append(a[i])
    y_0 = fourier_transform(a_0)
    y_1 = fourier_transform(a_1)
    y = []
    for k in range(int(n/2)):
        y.append(y_0[k] + w * y_1[k])
        w = w * w_n
    w = 1
```

```
    for k in range(int(n/2)):
        y.append(y_0[k] - w * y_1[k])
        w = w * w_n
    return y
```

ми створюємо нотний вектор

```
def note_vector(Pxx, f, avg=False, threshold=0):
    global notes_freq_list_dict, notes_list
    local_extrema_pxx = [PXX(i, Pxx[i]) for i in argrelextrema(Pxx,
np.greater)[0]]
    freqs_of_l_e_pxx = [FREQUENCY(pxx.position, f[pxx.position]) for pxx in
local_extrema_pxx]
    freqs_of_l_e_pxx_final = []
    temp = 0
    for i, frq in enumerate(freqs_of_l_e_pxx):
        if i == 0:
            temp = frq.value
            freqs_of_l_e_pxx_final.append(frq)
        else:
            if not hz_to_note(frq.value) == hz_to_note(temp):
                temp = frq.value
                freqs_of_l_e_pxx_final.append(frq)
    notes_of_l_e_pxx_final = [hz_to_note(frq.value) for frq in
freqs_of_l_e_pxx_final]
    n_d = dict.fromkeys(notes_list, 0)
    n_d_counter_dict = dict.fromkeys(notes_list, 0)
    for i, n in enumerate(notes_of_l_e_pxx_final):
        pxx_value = 0
        for j in local_extrema_pxx:
            if j.position == freqs_of_l_e_pxx_final[i].position:
                pxx_value = j.value
        n_d[n[:-1]] += pxx_value
        n_d_counter_dict[n[:-1]] += 1
    n_d_values = list(n_d.values())
    n_d_counter = list(n_d_counter_dict.values())
    if avg:
        for i, val in enumerate(n_d_values):
            try:
                n_d_values[i] = val / n_d_counter[i]
            except ZeroDivisionError:
                continue
    n_d_max_index = np.where(n_d_values == np.amax(n_d_values))[0][0]
    result_temp = [value/n_d_values[n_d_max_index] for value in n_d_values]
    result = [x if x > threshold else 0 for x in result_temp]
    return result
```

за допомогою якого ми визначаємо який це акорд

```
def notes_to_chords(n_v):
    global chords_list, notes_list, chord_matrix
    result = []
    for i in range(len(chords_list)):
        chord_matrix_vector = [x[i] for x in chord_matrix]
        cos_sim = np.dot(n_v,
chord_matrix_vector)/(np.linalg.norm(n_v)*np.linalg.norm(chord_matrix_vector)
)
        result.append(CHORD(chords_list[i], n_v, chord_matrix_vector,
cos_sim))
    return sorted(result, reverse=True, key=lambda x: x.confidence)
```

де `confidence`, це косинус кута між нотним вектором з матриці акордів та вектором, який ми отримали за допомогою функції `note_vector`. Косинус кута ми шукаємо як

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

де:

- $A \cdot B$ – скалярний добуток векторів,
- $\|A\|\|B\|$ – добуток норм векторів.

На вхід програми, повний код якої є у додатку А, йде WAV аудіофайл, який ми ділимо на рівні кадри певної довжини (яка задається користувачем); на виході отримуємо послідовність акордів та секунду, коли вони починають грати, та, за бажанням, WAV аудіофайл, сконструйований використовуючи послідовність акордів та функцію синус, та графіки коливань тиску повітря у вхідному та вихідному файлах. Приклад виводу програми можна побачити у додатках Б та В.

Висновок

Хоча розпізнавання та обробка музики є достатньо молодою дисципліною, порівняно з розпізнаванням мови, і на сьогоднішній день повне та точне розпізнавання оркестрової симфонії є практично неможливим завданням, проте ця галузь дуже швидко розвивається, та вже на сьогоднішній день ми не можемо уявити свого життя без неї.

Програми створюють музику, наприклад Logic Pro X, програми шукають музику, наприклад Shazam, програми музику відтворюють (прикладів безліч). Все це відбувається за допомогою математичних структур та алгоримів, математиків та програмістів, та підтримується такими спільнотами як ISMIR.

Програма, яку ми створили, дозволить професійним музикантам та початківцям розібрати пісню на акорди, що допоможе їм у їх творчих починаннях.

Список літератури

1. Meinard Müller, Fundamentals of music processing, Springer International Publishing Switzerland 2015, ISBN 978-3-319-21945-5
2. Андрущак Григорій, Застосування перетворення Фур'є на прикладі музики, Національний університет «Києво-Могилянська академія», Україна, Київ, 2019
3. Відповідність нот частотам [Електронний документ] – Режим доступу: URL: https://gist.github.com/i-Robi/8684800

Додаток А

Код програми по розпізнаванню акордів

```
import numpy.fft as fft
from matplotlib import pyplot as plt
import soundfile as sf
import numpy as np
from scipy.signal import argrelextrema
import os
import csv
from copy import deepcopy as dc


global chords_list, notes_list, chord_matrix, notes_dict,
notes_freq_list_dict
chord_matrix_csv_list = list(csv.reader(open("chord_matrix.csv", "r"),
delimiter=","))
chords_list = chord_matrix_csv_list[0][0].split(';')[1:]
notes_list = [row[0].split(';')[0] for row in chord_matrix_csv_list[1:]]
chord_matrix = np.array([i[0].split(';')[1:] for i in
chord_matrix_csv_list[1:]]).astype('int')
notes_dict = {
    16: 'C0',
    17: 'Db0',
    18: 'D0',
    19: 'Eb0',
    20: 'E0',
    21: 'F0',
    23: 'Gb0',
    24: 'G0',
    25: 'Ab0',
    27: 'A0',
    29: 'Bb0',
    30: 'B0',
    32: 'C1',
    34: 'Db1',
    36: 'D1',
    38: 'Eb1',
    41: 'E1',
    43: 'F1',
    46: 'Gb1',
    49: 'G1',
    51: 'Ab1',
    55: 'A1',
    58: 'Bb1',
    61: 'B1',
```

```
65: 'C2',
69: 'Db2',
73: 'D2',
77: 'Eb2',
82: 'E2',
87: 'F2',
92: 'Gb2',
98: 'G2',
103: 'Ab2',
110: 'A2',
116: 'Bb2',
123: 'B2',
130: 'C3',
138: 'Db3',
146: 'D3',
155: 'Eb3',
164: 'E3',
174: 'F3',
185: 'Gb3',
196: 'G3',
207: 'Ab3',
220: 'A3',
233: 'Bb3',
246: 'B3',
261: 'C4',
277: 'Db4',
293: 'D4',
311: 'Eb4',
329: 'E4',
349: 'F4',
369: 'Gb4',
392: 'G4',
415: 'Ab4',
440: 'A4',
466: 'Bb4',
493: 'B4',
523: 'C5',
554: 'Db5',
587: 'D5',
622: 'Eb5',
659: 'E5',
698: 'F5',
739: 'Gb5',
783: 'G5',
830: 'Ab5',
880: 'A5',
932: 'Bb5',
987: 'B5',
```

```
    1046: 'C6',
    1108: 'Db6',
    1174: 'D6',
    1244: 'Eb6',
    1318: 'E6',
    1396: 'F6',
    1479: 'Gb6',
    1567: 'G6',
    1661: 'Ab6',
    1760: 'A6',
    1864: 'Bb6',
    1975: 'B6',
    2093: 'C7',
    2217: 'Db7',
    2349: 'D7',
    2489: 'Eb7',
    2637: 'E7',
    2793: 'F7',
    2959: 'Gb7',
    3135: 'G7',
    3322: 'Ab7',
    3520: 'A7',
    3729: 'Bb7',
    3951: 'B7',
    4186: 'C8',
    4434: 'Db8',
    4698: 'D8',
    4978: 'Eb8'
}
notes_freq_list_dict = dict.fromkeys(notes_list, list())
for key, value in notes_dict.items():
    temp_list = dc(notes_freq_list_dict[str(value[:-1])])
    temp_list.append(key)
    notes_freq_list_dict[str(value[:-1])] = temp_list


class PXX:
    def __init__(self, position, value):
        self.value = value
        self.position = position

    def __repr__(self):
        return 'Pxx: v = {}, p = {}'.format(self.value, self.position)

class FREQUENCY:
    def __init__(self, position, value):
        self.value = value
        self.position = position
```

```
    def __repr__(self):
        return 'freq: v = {}, p = {}'.format(self.value, self.position)

class NOTE:
    def __init__(self, name, octave, frequency):
        self.name = name
        self.octave = octave
        self.frequency = frequency

class CHORD:
    def __init__(self, name, processed_note_vector, notes_vector,
confidence=1):
        self.name = name
        self.notes_vector = notes_vector
        self.processed_note_vector = processed_note_vector
        self.confidence = confidence

class CHORD_WITH_TIMING:
    def __init__(self, name, notes_vector, timing):
        self.name = name
        self.notes_vector = notes_vector
        self.timing = timing

    def __repr__(self):
        return 'Time: {} second, Chord: {}'.format(self.timing, self.name)

    def __str__(self):
        return 'Time: {} second, Chord: {}'.format(self.timing, self.name)


def hz_to_note(hz):
    global notes_dict
    return notes_dict.get(hz, notes_dict[min(notes_dict.keys(), key=lambda k:
abs(k-hz))])


def note_vector(Pxx, f, avg=False, threshold=0):
    global notes_freq_list_dict, notes_list
    local_extrema_pxx = [PXX(i, Pxx[i]) for i in argrelextrema(Pxx,
np.greater)[0]]
    freqs_of_l_e_pxx = [FREQUENCY(pxx.position, f[pxx.position]) for pxx in
local_extrema_pxx]
    freqs_of_l_e_pxx_final = []
    temp = 0
    for i, frq in enumerate(freqs_of_l_e_pxx):
        if i == 0:
            temp = frq.value
```

```
            freqs_of_l_e_pxx_final.append(frq)
        else:
            if not hz_to_note(frq.value) == hz_to_note(temp):
                temp = frq.value
                freqs_of_l_e_pxx_final.append(frq)
    notes_of_l_e_pxx_final = [hz_to_note(frq.value) for frq in
freqs_of_l_e_pxx_final]
    n_d = dict.fromkeys(notes_list, 0)
    n_d_counter_dict = dict.fromkeys(notes_list, 0)
    for i, n in enumerate(notes_of_l_e_pxx_final):
        pxx_value = 0
        for j in local_extrema_pxx:
            if j.position == freqs_of_l_e_pxx_final[i].position:
                pxx_value = j.value
        n_d[n[:-1]] += pxx_value
        n_d_counter_dict[n[:-1]] += 1
    n_d_values = list(n_d.values())
    n_d_counter = list(n_d_counter_dict.values())
    if avg:
        for i, val in enumerate(n_d_values):
            try:
                n_d_values[i] = val / n_d_counter[i]
            except ZeroDivisionError:
                continue
    n_d_max_index = np.where(n_d_values == np.amax(n_d_values))[0][0]
    result_temp = [value/n_d_values[n_d_max_index] for value in n_d_values]
    result = [x if x > threshold else 0 for x in result_temp]
    return result


def notes_to_chords(n_v):
    global chords_list, notes_list, chord_matrix
    result = []
    for i in range(len(chords_list)):
        chord_matrix_vector = [x[i] for x in chord_matrix]
        cos_sim = np.dot(n_v,
chord_matrix_vector)/(np.linalg.norm(n_v)*np.linalg.norm(chord_matrix_vector)
)
        result.append(CHORD(chords_list[i], n_v, chord_matrix_vector,
cos_sim))
    return sorted(result, reverse=True, key=lambda x: x.confidence)


def gauss_w(n, frame):
    Q = 0.5
    a = (frame - 1) / 2
    t = (n - a) / (Q * a)
    t = t * t
```

```
    return np.exp(-t / 2)


def fourier_transform(a):
    n = len(a)
    if n == 1:
        return a
    if n & (n - 1):
        raise ValueError('array length must be a power of 2')
    w_n = np.e**((2*np.pi*1j)/n)
    w = 1
    a_0 = []
    a_1 = []
    for i in range(n):
        if i % 2 == 0:
            a_0.append(a[i])
        else:
            a_1.append(a[i])
    y_0 = fourier_transform(a_0)
    y_1 = fourier_transform(a_1)
    y = []
    for k in range(int(n/2)):
        y.append(y_0[k] + w * y_1[k])
        w = w * w_n
    w = 1
    for k in range(int(n/2)):
        y.append(y_0[k] - w * y_1[k])
        w = w * w_n
    return y


def make_chord_wav(note_vector, duration, sample_rate, octave=3,
show_plot=True, chord_name=None):
    global notes_freq_list_dict
    Fs = sample_rate
    T = 1 / Fs
    t = duration
    N = Fs * t
    freqs = [note[octave] for note in list(notes_freq_list_dict.values())]
    omegas = [2 * np.pi * freq for freq in freqs]
    t_vec = np.arange(N) * T
    y_list = [np.sin(omega * t_vec) * note_vector[i] for i, omega in
enumerate(omegas)]
    y = y_list[0]
    for y_l_e in y_list[1:]:
        y = np.add(y, y_l_e)
    y_max_index = np.where(y == np.amax(y))[0][0]
    y = [value/y[y_max_index] for value in y]
```

```
    if show_plot:
        try:
            plt.plot(t_vec[:1200], y[:1200])
        except ValueError as e:
            err_vals = [int(e.args[0].split()[-3][1:-2]),
int(e.args[0].split()[-1][1:-2])]
            print(err_vals)
            if err_vals[0] > err_vals[1]:
                plt.plot(t_vec[:(err_vals[1] - err_vals[0])], y)
            else:
                plt.plot(t_vec, y[:(err_vals[0] - err_vals[1])])
        plt.ylabel('Amplitude')
        plt.xlabel('Time (seconds)')
        plt.title(str(chord_name) + ' chord, Octave: ' + str(octave))
        plt.show()
        if not os.path.exists('out/'):
            os.makedirs('out/')
    sf.write('out/'+str(chord_name)+'_chord_'+str(octave)+'_octave.wav', y,
sample_rate)


def make_song_wav(chords, sample_rate, filename, file_length, octave=4,
show_plot=False):
    global notes_freq_list_dict
    y_res = list()
    Fs = sample_rate
    T = 1 / Fs
    N_final = Fs * file_length
    t_vec_final = np.arange(N_final) * T
    for chord_i, chord in enumerate(chords):
        if chord_i == len(chords) - 1:
            t = file_length - chord.timing
        else:
            t = chords[chord_i+1].timing - chord.timing
        N = Fs * t
        freqs = [note[octave] for note in
list(notes_freq_list_dict.values())]
        omegas = [2 * np.pi * freq for freq in freqs]
        t_vec = np.arange(N) * T
        y_list = [np.sin(omega * t_vec) * chord.notes_vector[i] for i, omega
in enumerate(omegas)]
        y = y_list[0]
        for y_l_e in y_list[1:]:
            y = np.add(y, y_l_e)
        y_res.extend(y)
    y_max_index = np.where(y_res == np.amax(y_res))[0][0]
    y_final_result = [value/y_res[y_max_index] for value in y_res]
    if show_plot:
```

```
        try:
            plt.plot(t_vec_final[:1200], y_final_result[:1200])
        except ValueError as e:
            err_vals = [int(e.args[0].split()[-3][1:-2]),
int(e.args[0].split()[-1][1:-2])]
            print(err_vals)
            if err_vals[0] > err_vals[1]:
                plt.plot(t_vec_final[:(err_vals[1] - err_vals[0])],
y_final_result)
            else:
                plt.plot(t_vec_final, y_final_result[:(err_vals[0] -
err_vals[1])])
        plt.ylabel('Amplitude')
        plt.xlabel('Time (seconds)')
        plt.title(filename)
        plt.show()
        if not os.path.exists('out/'):
            os.makedirs('out/')
    sf.write('out/'+filename+'.wav', y_final_result, sample_rate)


def process_sound(filename, t_start=0, t_finish=None, power_of_two=None,
gauss_window=True, show_plots=False, return_wav=False, debug=False,
file_info=False, bf=False, avg=False, threshold=0):
    data, samplerate = sf.read(filename)
    if not (power_of_two is None):
        data = data[int(t_start * samplerate):int(t_start*samplerate +
2**power_of_two)]
    else:
        if t_finish is None:
            data = data[int(t_start * samplerate):]
        else:
            data = data[int(t_start * samplerate):int(t_finish * samplerate)]
    T = 1 / samplerate
    t = len(data) / samplerate
    N = samplerate * t
    if file_info:
        print('Samplerate:', str(samplerate))
        print('Data length:', str(len(data)))
        print('Sampling period:', str(T))
        print('Duration (seconds):', str(t))
    if debug:
        print('Data:\n', data)
    t_vec = np.arange(N) * T
    try:
        y = np.array([x[0] for x in data])
    except IndexError as e:
        y = data
```

```
    if show_plots:
        try:
            plt.plot(t_vec, y)
        except ValueError as e:
            err_vals = [int(e.args[0].split()[-3][1:-2]),
int(e.args[0].split()[-1][1:-2])]
            if err_vals[0] > err_vals[1]:
                plt.plot(t_vec[:(err_vals[1]-err_vals[0])], y)
            else:
                plt.plot(t_vec, y[:(err_vals[0] - err_vals[1])])
        plt.ylabel('Amplitude')
        plt.xlabel('Time (seconds)')
        plt.show()
    try:
        Y_k = np.array(fourier_transform(y))[0:int(N / 2)] / N
    except ValueError as ve:
        Y_k = fft.fft(y)[0:int(N / 2)] / N
    Y_k = 2 * Y_k
    Pxx_temp = np.abs(Y_k)
    if gauss_window:
        Pxx_temp1 = [Pxx_temp[i] * gauss_w(i, len(Pxx_temp)) for i in
range(len(Pxx_temp))]
        Pxx = np.array(Pxx_temp1)
    else:
        Pxx = np.array(Pxx_temp)
    f = samplerate * np.arange((N / 2)) / N
    chord = notes_to_chords(note_vector(Pxx, f, avg=avg,
threshold=threshold))[0]
    if return_wav:
        make_chord_wav(chord.notes_vector, 2, samplerate,
chord_name=chord.name, show_plot=show_plots)
    return chord


def process_song(filename, part_length, power_of_two=None, gauss_window=True,
show_plots=False, return_wav=False, debug=False, file_info=False, bf=False,
avg=False, threshold=0):
    data, samplerate = sf.read(filename)
    t = len(data) / samplerate
    res_chords = []
    for i in range(int(t/part_length)):
        print(i)
        start = i * part_length
        finish = start + part_length
        chord = process_sound(filename, start, finish, show_plots=show_plots,
return_wav=False, file_info=file_info, gauss_window=gauss_window,
debug=debug, avg=avg, threshold=threshold)
        if len(res_chords) == 0:
```

```
            res_chords.append(CHORD_WITH_TIMING(chord.name,
chord.notes_vector, start))
        elif not chord.name == res_chords[-1].name or len(res_chords) == 0:
            res_chords.append(CHORD_WITH_TIMING(chord.name,
chord.notes_vector, start))
    if return_wav:
        make_song_wav(res_chords, samplerate, filename.split('/')[-
1].split('.')[0], t)
    return res_chords


if __name__ == '__main__':
    chords = process_song('samples/the_beatles_let_it_be.wav', 1,
return_wav=True)
    print(chords)
```

Додаток Б

Вивід програми на прикладі аудіофайлу the_beatles_let_it_be.wav

[Time: 0 second, Chord: F, Time: 1 second, Chord: G#, Time: 2 second, Chord: Cm, Time: 3 second, Chord: C, Time: 4 second, Chord: A, Time: 5 second, Chord: F, Time: 7 second, Chord: Fm, Time: 8 second, Chord: Cm, Time: 9 second, Chord: G, Time: 10 second, Chord: F, Time: 11 second, Chord: Fm, Time: 12 second, Chord: C, Time: 13 second, Chord: G, Time: 14 second, Chord: Fm, Time: 15 second, Chord: Cm, Time: 16 second, Chord: D, Time: 17 second, Chord: Am, Time: 18 second, Chord: Dm, Time: 19 second, Chord: F, Time: 20 second, Chord: C, Time: 22 second, Chord: G, Time: 23 second, Chord: F, Time: 24 second, Chord: C, Time: 27 second, Chord: D#, Time: 28 second, Chord: G, Time: 29 second, Chord: D, Time: 31 second, Chord: G#, Time: 32 second, Chord: C, Time: 33 second, Chord: Fm, Time: 34 second, Chord: G, Time: 35 second, Chord: Cm, Time: 36 second, Chord: F, Time: 37 second, Chord: C, Time: 38 second, Chord: Am, Time: 40 second, Chord: G, Time: 41 second, Chord: Gm, Time: 42 second, Chord: Cm, Time: 43 second, Chord: E, Time: 44 second, Chord: C#m, Time: 45 second, Chord: C, Time: 46 second, Chord: Fm, Time: 47 second, Chord: G, Time: 48 second, Chord: Cm, Time: 49 second, Chord: F, Time: 50 second, Chord: Fm, Time: 51 second, Chord: C, Time: 52 second, Chord: Cm, Time: 53 second, Chord: Gm, Time: 54 second, Chord: G, Time: 55 second, Chord: Em, Time: 56 second, Chord: Dm, Time: 57 second, Chord: F, Time: 58 second, Chord: C, Time: 60 second, Chord: G, Time: 61 second, Chord: Cm, Time: 62 second, Chord: C, Time: 63 second, Chord: Fm, Time: 64 second, Chord: Cm, Time: 65 second, Chord: G, Time: 67 second, Chord: Cm, Time: 68 second, Chord: Am, Time: 69 second, Chord: Em, Time: 70 second, Chord: Dm, Time: 71 second, Chord: F, Time: 72 second, Chord: C, Time: 74 second, Chord: G, Time: 75 second, Chord: F, Time: 76 second, Chord: C, Time: 77 second, Chord: Cm, Time: 78 second, Chord: G#m, Time: 79 second, Chord: Am, Time: 80 second, Chord: D#, Time: 81 second, Chord: Cm, Time: 82 second, Chord: C#, Time: 83 second, Chord: C, Time: 86 second, Chord: Am, Time:

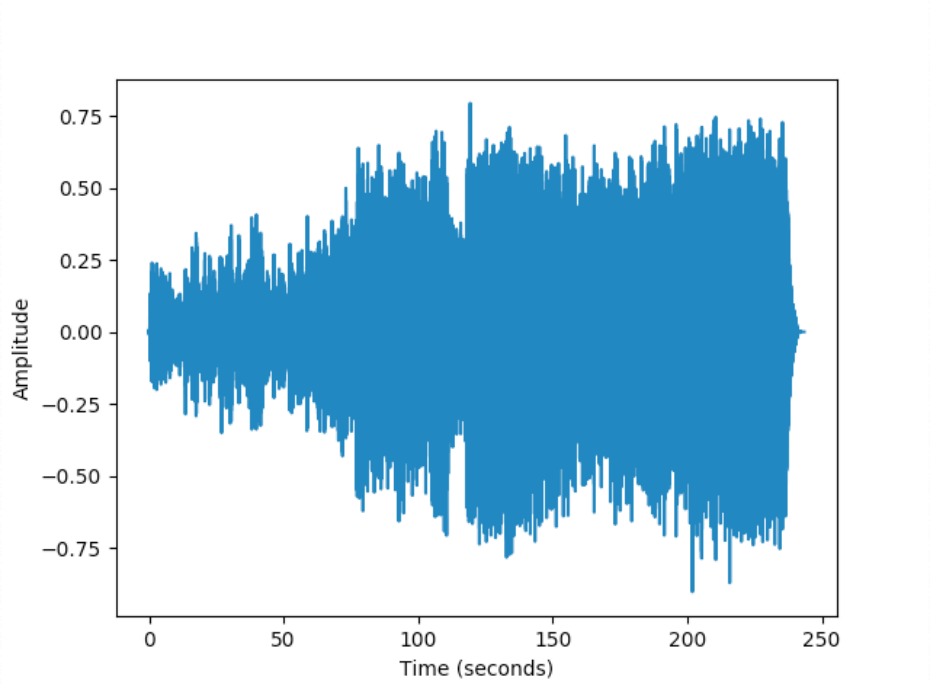
87 second, Chord: Cm, Time: 88 second, Chord: A#, Time: 89 second, Chord: Dm, Time: 90 second, Chord: Cm, Time: 91 second, Chord: G#, Time: 92 second, Chord: A#m, Time: 93 second, Chord: D#, Time: 94 second, Chord: Cm, Time: 96 second, Chord: C, Time: 100 second, Chord: Cm, Time: 101 second, Chord: G, Time: 102 second, Chord: Dm, Time: 103 second, Chord: F, Time: 104 second, Chord: Cm, Time: 105 second, Chord: D#m, Time: 106 second, Chord: A#, Time: 107 second, Chord: G#, Time: 108 second, Chord: D#, Time: 109 second, Chord: A#, Time: 110 second, Chord: C, Time: 112 second, Chord: F, Time: 113 second, Chord: Fm, Time: 115 second, Chord: Gm, Time: 116 second, Chord: C, Time: 117 second, Chord: Cm, Time: 118 second, Chord: A#m, Time: 119 second, Chord: Cm, Time: 120 second, Chord: C, Time: 121 second, Chord: Gm, Time: 122 second, Chord: D#, Time: 123 second, Chord: A, Time: 124 second, Chord: Dm, Time: 125 second, Chord: A#m, Time: 126 second, Chord: C, Time: 127 second, Chord: Cm, Time: 128 second, Chord: D#, Time: 129 second, Chord: F, Time: 130 second, Chord: Dm, Time: 131 second, Chord: Cm, Time: 132 second, Chord: C, Time: 133 second, Chord: Cm, Time: 134 second, Chord: C, Time: 135 second, Chord: G, Time: 136 second, Chord: A#, Time: 137 second, Chord: D#, Time: 138 second, Chord: A#, Time: 139 second, Chord: A#m, Time: 140 second, Chord: C, Time: 142 second, Chord: Cm, Time: 143 second, Chord: Em, Time: 144 second, Chord: Dm, Time: 145 second, Chord: Cm, Time: 146 second, Chord: Am, Time: 148 second, Chord: D#, Time: 149 second, Chord: C, Time: 150 second, Chord: F#m, Time: 151 second, Chord: A#, Time: 152 second, Chord: Am, Time: 153 second, Chord: A#m, Time: 154 second, Chord: F, Time: 156 second, Chord: G, Time: 157 second, Chord: F, Time: 158 second, Chord: Dm, Time: 159 second, Chord: C, Time: 160 second, Chord: Am, Time: 161 second, Chord: Gm, Time: 162 second, Chord: Cm, Time: 163 second, Chord: G, Time: 164 second, Chord: D#, Time: 165 second, Chord: F, Time: 166 second, Chord: D#m, Time: 167 second, Chord: D, Time: 168 second, Chord: Am, Time: 169 second, Chord: D#, Time: 170 second, Chord: Cm, Time: 171 second, Chord: C#, Time: 172 second, Chord: Dm, Time: 173 second, Chord: A, Time: 174 second, Chord: G#, Time: 175 second, Chord: Cm, Time: 176 second,

Chord: G, Time: 178 second, Chord: Dm, Time: 179 second, Chord: Fm, Time: 180 second, Chord: Dm, Time: 181 second, Chord: C#, Time: 182 second, Chord: C, Time: 184 second, Chord: Cm, Time: 185 second, Chord: A#m, Time: 186 second, Chord: Dm, Time: 187 second, Chord: G#, Time: 188 second, Chord: A#m, Time: 189 second, Chord: A, Time: 191 second, Chord: Em, Time: 192 second, Chord: Dm, Time: 194 second, Chord: Am, Time: 197 second, Chord: Cm, Time: 198 second, Chord: C#m, Time: 199 second, Chord: A#m, Time: 200 second, Chord: F, Time: 202 second, Chord: A, Time: 203 second, Chord: Am, Time: 204 second, Chord: A, Time: 205 second, Chord: C, Time: 206 second, Chord: Dm, Time: 207 second, Chord: D, Time: 208 second, Chord: C, Time: 209 second, Chord: Am, Time: 211 second, Chord: D#, Time: 212 second, Chord: G, Time: 213 second, Chord: Dm, Time: 214 second, Chord: A, Time: 215 second, Chord: C, Time: 216 second, Chord: Dm, Time: 217 second, Chord: A, Time: 218 second, Chord: D#, Time: 219 second, Chord: Em, Time: 220 second, Chord: Dm, Time: 221 second, Chord: A#, Time: 222 second, Chord: C, Time: 223 second, Chord: D#, Time: 224 second, Chord: C, Time: 225 second, Chord: Gm, Time: 226 second, Chord: G, Time: 227 second, Chord: Dm, Time: 228 second, Chord: F, Time: 229 second, Chord: C, Time: 230 second, Chord: F, Time: 231 second, Chord: C#, Time: 232 second, Chord: Cm, Time: 233 second, Chord: C, Time: 234 second, Chord: G#m, Time: 235 second, Chord: F, Time: 236 second, Chord: C, Time: 239 second, Chord: Cm, Time: 240 second, Chord: C]

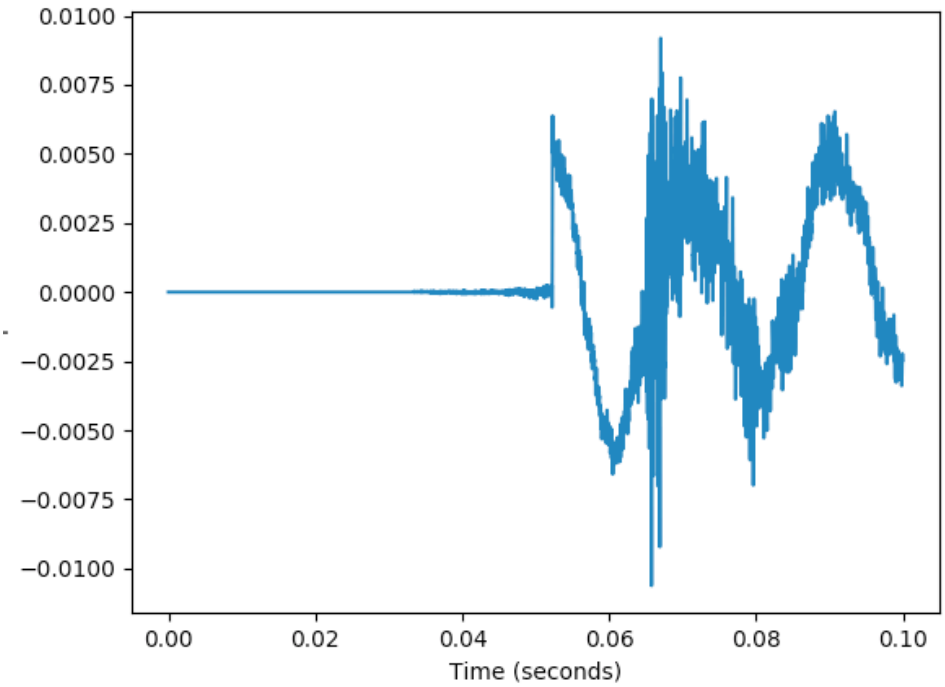
Додаток В

Графіки коливань тиску повітря у вхідному та вихідному файлах на прикладі аудіофайлу the_beatles_let_it_be.wav

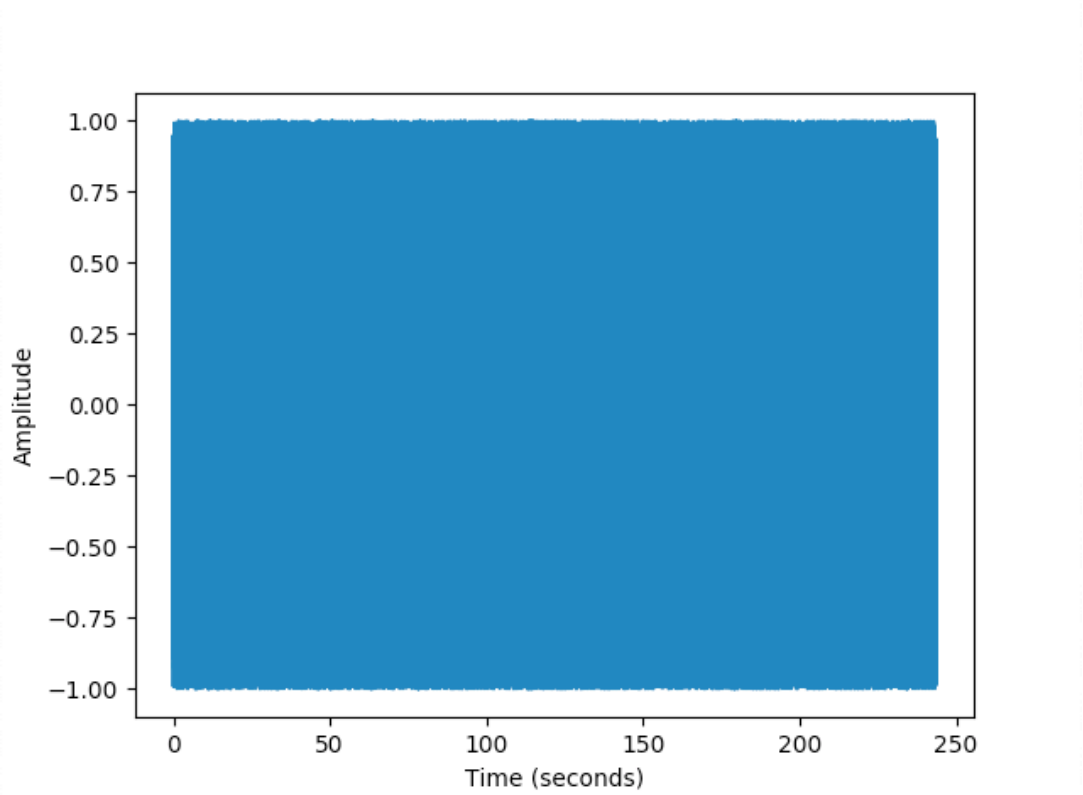
Вхідний файл:



Вхідний файл (перші 0,1 секунди):



Вихідний файл:



Вихідний файл (перші 0,1 секунди):