

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа киберфизических систем и управления

Работа допущена к защите

Руководитель ОП

_____ А.А. Ефремов

«___» _____ 20__ г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

ТРАНСКРИПЦИЯ МУЗЫКАЛЬНОГО ПРОИЗВЕДЕНИЯ С ИСПОЛЬЗОВАНИЕМ ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЙ

по направлению подготовки (специальности) 09.03.02 Информационные
системы и технологии

Направленность (профиль) 09.03.02_02 Информационные системы и
технологии

Выполнил
студент гр. 3530902/70201



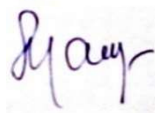
Т.И. Матченко

Руководитель
доцент, к.т.н,



С.В. Хлопин

Консультант
по нормоконтролю



В.Е. Маер

Санкт-Петербург – 2021

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО
Институт компьютерных наук и технологий
Высшая школа киберфизических систем и управления

УТВЕРЖДАЮ

Руководитель ОП

_____ А.А.Ефремов

« ____ » _____ 2021г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Матченко Татьяне Ивановне, гр. 3530902/70201

1. Тема выпускной работы: Транскрипция музыкального произведения с использованием вейвлет-преобразований.
2. Срок сдачи студентом законченной работы: 07.06.2021
3. Исходные данные по преддипломной практике: Методические рекомендации по подготовке ВКР; материалы сайтов библиотек российских университетов; справочная документация к программному обеспечению; литература по теме ВКР.
4. Содержание работы (перечень подлежащих разработке вопросов): работа содержит пять глав. В первой главе рассмотрены понятия, связанные с транскрипцией музыкального сигнала, поставлена задача. Во второй главе были рассмотрены возможные алгоритмы для решения задачи, был выбран один из них. В третьей главе рассмотрена теория применения выбранного метода в данной задаче. В четвертой главе реализован алгоритм применения метода. В пятой главе оценено качество работы программы.
5. Перечень графического материала (с указанием обязательных чертежей): презентация к докладу на защите выпускной работы.
6. Консультанты по работе:
7. Дата выдачи задания: 05.05.2020

Руководитель ВКР _____

(подпись)

С.В. Хлопин

Задание принял к исполнению 18.01.2021

Студент _____

(подпись)

Т. И. Матченко

РЕФЕРАТ

На 62 страницы, 25 рисунков, 8 источников, 9 приложений.

КЛЮЧЕВЫЕ СЛОВА: ТРАНСКРИПЦИЯ, ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ, МУЗЫКА, СИГНАЛ, НОТА, РАСПОЗНАВАНИЕ.

Тема выпускной квалификационной работы: «Транскрипция музыкального произведения с использованием вейвлет-преобразований».

Данная работа посвящена программной транскрипции музыкального файла.

В ходе работы решались следующие задачи:

1. Изучение литературы по тематике транскрипции музыкального произведения и методов программной транскрипции.
2. Реализация алгоритма непрерывного вейвлет-преобразования и транскрипции музыкального файла.
3. Оценка качества работы реализованного алгоритма.

В результате был изучен и применен алгоритм непрерывного вейвлет-преобразования.

ABSTRACT

62 pages, 25 figures, 6 tables, 9 appendices.

KEYWORDS: TRANSCRIPTION, WAVELET TRANSFORM, MUSIC, SIGNAL, NOTE, RECOGNITION.

The subject of the graduate qualification work is «Transcription of a musical composition with wavelet transform».

This work is devoted to the software transcription of a music file. In the course of the work, the following tasks were solved:

1. Study of the literature on the topic of transcription of a musical work and methods of software transcription.
2. Implementation of the algorithm for continuous wavelet transformation and transcription of a music file.
3. Evaluation of the quality of the implemented algorithm.

As a result, the algorithm of the continuous wavelet transform was studied and applied.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ	8
ГЛАВА 2. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ	10
2.1. Дискретное преобразование Фурье.....	10
2.2. Спектральные, временные и спектровременные методы	11
2.2.1. Спектральные методы.....	11
2.2.2. Временные методы.....	12
2.2.3. Спектрально-временные методы	12
2.3. Мультичастотная оценка и распознавание полифонии	13
2.4. Спектральный подход	13
2.5. Статистический подход.....	14
2.6. Методы разложения спектрограммы	15
2.7. Методы с обучением, полуавтоматический подход.....	15
2.8. Применение генетических алгоритмов.....	16
2.9. Выбор метода	18
ГЛАВА 3. НЕПРЕРЫВНОЕ ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ В ЗАДАЧЕ ТРАНСКРИПЦИИ СИГНАЛА	20
ГЛАВА 4. ПРИМЕНЕНИЕ НЕПРЕРЫВНОГО ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЯ.....	24
4.1. Синусоидальный сигнал.....	24
4.1.1. Получение сигнала.....	24
4.1.2. Создание вейвлета.....	25
4.1.3. Применение НВП	25
4.1.4. Обработка данных.....	26
4.1.5. Поиск нотных объектов.....	27
4.1.6. Применение на другом примере	28
4.2. Сигнал, приближенный к реальному	30
4.2.1. Получение сигнала.....	30
4.2.2. Создание вейвлета.....	31
4.2.3. Применение НВП	33
4.2.4. Обработка данных.....	34
4.2.5. Нахождение нотных объектов	35
4.2.6. Об использовании подходящего вейвлета.....	37
ГЛАВА 5. ОЦЕНКА РАБОТЫ ТРАНСКРИПЦИИ	40

5.1. Характеристики оценки.....	40
5.2. Анализ работы алгоритма	41
ЗАКЛЮЧЕНИЕ.....	46
СПИСОК ЛИТЕРАТУРЫ.....	48
ПРИЛОЖЕНИЕ 1 Код программы для непрерывного вейвлет-преобразования.....	49
ПРИЛОЖЕНИЕ 2 Код программы для эксперимента с синусоидальным сигналом	50
ПРИЛОЖЕНИЕ 3 Код собственных фильтров, использованных в программе	52
ПРИЛОЖЕНИЕ 4 Код программы для поиска нотных объектов в результатах НВП синусоидального сигнала.....	54
ПРИЛОЖЕНИЕ 5 Код программы для эксперимента с сигналом, приближенным к реальному	55
ПРИЛОЖЕНИЕ 6 Код программы для поиска нотных объектов в результатах НВП сигнала, приближенного к реальному	57
ПРИЛОЖЕНИЕ 7 Код программы для записи распознанных нот в MIDI файл и сравнения с исходным MIDI файлом.....	59

ВВЕДЕНИЕ

Музыкальная транскрипция — это преобразование необработанного музыкального сигнала в символьное представление.

Традиционно музыкальная нотация указывает высоту тона, выбранный инструмент, время и длительность каждого воспроизводимого звука. Цель транскрипции музыки состоит в том, чтобы вывести эти музыкальные параметры, имея в виду только акустическую запись исполнения. С точки зрения представления данных это можно рассматривать как преобразование аудиосигнала в файл MIDI. Особый интерес здесь представляют сигналы полифонической музыки, в которых одновременно звучит несколько звуков.

Автоматическое восстановление нотной записи аудиосигнала позволяет модифицировать, перестраивать и обрабатывать музыку на высоком уровне абстракции, а затем повторно синтезировать ее. Еще одним важным приложением является сжатие звука: например, представление в виде MIDI-файла чрезвычайно компактно, но в значительной степени сохраняет характеристики музыкального произведения. Музыкальную транскрипцию можно использовать в таких областях, как поиск информации, музыкальный анализ импровизированной и этнической музыки, а также в интерактивных музыкальных системах, которые создают аккомпанемент к пению или игре солиста.

Такие задачи требуют разработки средств транскрипции музыки, которые позволят получать данные в автоматическом режиме.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ

Поставленная задача заключается в том, чтобы обработать музыкальный файл и получить партитуру для инструмента, то есть перечислить сыгранные ноты и их характеристики: время начала, длительность, громкость. Партитуру представить в удобном для восприятия виде.

Математическая модель музыкального сигнала может быть представлена в виде следующей формулы [6, с.1]:

$$F(t) = h(t) + \sum_{i=1}^k A_i n_i(t - \theta_i), \quad (1.1)$$

где F – музыкальный сигнал одного инструмента;

k – количество сыгранных нот;

A_i – громкость звучания каждой ноты;

n_i – амплитудно-временные характеристики отдельных нот мелодии;

θ_i – время начала звучания каждой ноты;

$h(t)$ – сигнал помехи при звукозаписи.

Соответственно, задача восстановления партитуры может быть сформулирована так: для заданного сигнала $F(t)$ и заданных амплитудно-временных характеристик допустимых нот $N_1 \dots N_j$ определить для всех i следующие величины: $A_i, \theta_i, n_i \in N_1 \dots N_j$.

Большинство музыкальных инструментов позволяют получить амплитудно-временные характеристики нот с помощью масштабирования такой характеристики одной заданной ноты:

$$N_i(t) = N_0 \left(\frac{t}{m_i} \right), \quad (1.2)$$

где N_i — амплитудно-временные характеристики ноты i ;

N_0 — амплитудно-временные характеристики базовой ноты;

i — положение ноты $N_i(t)$ по высоте относительно ноты $N_0(t)$;

m_i — коэффициент масштабирования для ноты i .

Для большинства инструментов характерен равномерно-темперированный строй, то есть коэффициенты m_i обычно равны $2^{-i/12}$, где $i \in \mathbb{Z}$. Например, если рассматривать ноту "до" первой октавы как базовую (N_0), то характеристика ноты "до" следующей октавы может представлена как $N_{12}(t) = N_0\left(\frac{t}{2^{12/12}}\right)$. В качестве базовой ноты может быть выбрана любая нота из диапазона инструмента.

Таким образом, задача может быть переформулирована почти так же, с отличием в том, что теперь нам не обязательно обладать характеристиками вообще всех нот инструмента: достаточно иметь только характеристику базовой ноты.

Таким образом, задача для одного инструмента формулируется так: для заданного сигнала $F(t)$ найти

$$\arg \min_{m_1 \dots m_k, A_1 \dots A_k, \theta_1 \dots \theta_k} \left\| \sum_{i=1}^k A_i \cdot N_0\left(\frac{t - \theta_i}{m_i}\right) - F(t) \right\|, \quad (1.3)$$

где k – количество распознанных нот;

A_i – громкость звучания каждой ноты;

N_0 – амплитудно-временные характеристики базовой ноты;

m_i – коэффициент масштабирования для каждой ноты;

θ_i – время начала звучания каждой ноты.

Задачу для нескольких инструментов можно сформулировать аналогично. Нужно лишь выяснить базовые ноты для каждого инструмента. Тогда можно сформулировать задачу так:

$$\arg \min_{m_{1,1} \dots m_{l,k}, A_{1,1} \dots A_{l,k}, \theta_{1,1} \dots \theta_{l,k}} \left\| \sum_{j=1}^l \sum_{i=1}^k A_{j,i} \cdot N_0\left(\frac{t - \theta_{j,i}}{m_{j,i}}\right) - F(t) \right\|, \quad (1.4)$$

где l – количество инструментов.

ГЛАВА 2. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

Для решения задачи распознавания музыкального звука, также называемой задачей Automatic Music Transcription, существуют различные подходы. Они будут рассмотрены в этой главе.

2.1. Дискретное преобразование Фурье

Для сжатия звука в MP3, изображений в JPEG и других алгоритмов цифровой обработки сигналов, а также анализа частот в дискретном сигнале используются модификации преобразования Фурье (Discrete Fourier Transform). Входом в дискретное преобразование Фурье служит дискретная функция.

Прямое преобразование Фурье:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn}, k = 0, 1, \dots, N-1 \quad (2.1)$$

Обратное преобразование Фурье:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn}, n = 0, 1, \dots, N-1, \quad (2.2)$$

где N – количество значений сигнала, измеренных за период;

x_n – измеренные значения сигнала;

X_k – комплексные амплитуды синусоидальных сигналов, слагающих исходный сигнал;

k – индекс частоты.

В результате преобразования сигнал раскладывается на гармоники (синусоидальные составляющие) с частотами от N до 1 колебаний за период. Величина частоты дискретизации, равная N отсчетов за период, является причиной возникновения муарового эффекта, так как падает точность отображения высокочастотных составляющих. В результате вторая половина из N комплексных амплитуд не несет дополнительной информации, так как фактически является зеркальным отражением первой. Это является причиной, по которой при обработке музыкального звука частота дискретизации берется

равной 44100 – числу, вдвое превышающему диапазон, который слышит человек. То есть при отсечении второй половины, не несущей смысловой нагрузки, получается весь слышимый диапазон.

Стоит упомянуть о быстром преобразовании Фурье — оптимизированном по скорости алгоритме вычисления дискретного преобразования Фурье. Для прямого вычисления дискретного преобразования Фурье требуется количество действий $O(N^2)$, для быстрого вычисления это число меньше. Иногда под быстрым преобразованием Фурье подразумевают алгоритм прореживания по частоте или времени, имеющий сложность $O(N \log(N))$.

2.2. Спектральные, временные и спектровременные методы

Существуют алгоритмы для решения проблемы распознавания одноголосых мелодий — мелодий, представленных сигналом с одной фундаментальной частотой. Их основное предположение – в одно время звук образован сигналом только одной гармоникой. Рассмотрим существующие подходы.

2.2.1. Спектральные методы

Гармоники звука появляются только на целых множителях фундаментальной частоты. Из этого следует, что на основе анализа спектра может быть сделано предположение о высоте звука. Автокорреляционная функция, максимум которой отвечает фундаментальной частоте для гармонического спектра, используется для определения повторяющихся шаблонов в сигнале.

Группа ученых — M.Lahat, R.Niederjohn и D.Krubsack — предложила на основе сглаживания спектра сигнала и оценки фундаментальной частоты, проведенной с помощью автокорреляционных функций, метод, определяющий высоту распознаваемой речи, в которой присутствует большая доля шума.

Имеет смысл упомянуть работу Брауна «Musical fundamental frequency tracking using a pattern recognition method», в ней он вычисляет спектр, представляющий спектр на логарифмической шкале. Числа, стоящие на позициях гармоник, составляют идеальный шаблон, который участвует в кросс-корреляции с логарифмическим спектром. В ходе вычисления последовательно определяется высота, причем максимум взаимной корреляционной функции определяет высоту на заданном участке временной оси. Поскольку расстояние между гармониками постоянно для всех высот, оказывается логичным использовать логарифмический вид гармонических шаблонов.

2.2.2. Временные методы

Автокорреляционная функция используется для анализа входящей волны. Это составляет базовый подход для анализа относительно времени.

Формула входящей волны:

$$ACF[v] = \frac{1}{N} \sum_{n=0}^{N-v-1} x[n]x[n+v] \quad (2.3)$$

Фундаментальный период периодической волны отражается первым пиком этой функции.

Однако, возможно возникновение субгармонических ошибок, когда пики появляются на множителях периода. Тем не менее, использование автокорреляционной функции эффективно из-за применения дискретного преобразования Фурье (кроме того, в литературе можно найти улучшенные варианты этой функции).

2.2.3. Спектрально-временные методы

Методы оценки высоты делятся на два вида: основанные на анализе спектра (в них периодически возникают гармонические ошибки – ошибки на целых множителях фундаментальной частоты) и временные, они, в свою очередь, выдают ошибки на субгармониках. Поэтому было принято решение найти алгоритмы, являющиеся средним между перечисленными выше двумя

группами. Такие алгоритмы можно было найти, например, разделив входящий сигнал на полосы, которые обрабатываются отдельными каналами. Для реализации этой идеи была создана унитарная модель Меддиса и Хьюитта. Она похожа на человеческое восприятие звука и включает в себя следующие шаги:

1. Проход входящего сигнала через логарифмический фильтр;
2. Сглаживание выхода каждого фильтра и применение к каждому каналу фильтра низких частот;
3. Применение автокорреляционной функции к каждому каналу;
4. Суммирование результатов.

2.3. Мультичастотная оценка и распознавание полифонии

Особенность полифонической музыки в том, что для ее распознавания требуется определять ноты, которые помимо одновременного появления могут быть исполнены разными инструментами. Таким образом, основной проблемой полифонии становится мультичастотное распознавание.

2.4. Спектральный подход

Спектральный подход порождает семейство мультичастотных методов, являющих собой большинство среди всех известных алгоритмов распознавания полифонии. Для преобразования сигнала исследователи использовали оконное преобразование Фурье, фильтрующие методы, constant-Q преобразование, вейвлет-преобразования и прочие. Было предложено разделить системы для полифонии на совместные и итеративные, которые извлекают самую подходящую высоту на каждом шаге. Из-за указанной особенности модели имеют тенденцию на каждом шаге накапливать ошибку, но при этом они дешевы в вычислительном плане, и этим выигрывают.

В отличие от итеративных совместные системы имеют высокую вычислительную стоимость, поскольку оценивают комбинации

обнаруживаемых фундаментальных частот, но полученные в результате оценки оказываются наиболее точными, поэтому многие исследователи предпочитают использовать совместную категорию.

Среди подходов, использующих различные способы спектрального разложения, следует отметить разложение неотрицательных матриц и методы максимального правдоподобия, последние используют expectation maximization алгоритм, оценивая спектральную огибающую возможных высот.

Обычно, системы, целью которых является определить несколько частот, используют методы, перенятые от обработки сигнала. Частотновременное представление входящего сигнала позволяет извлечь свойства, которые определяют ноту. Клапури в своей работе предложил метод, основная идея которого заключалась в том, что огибающая гармонического звука обладает гладкостью. Функция максимума способна справиться с негармоничностью, поэтому ее используют для определения доминирующей частоты. На следующем этапе вычисляется спектр полученного звука, она сглаживается и вычитается из общего спектра. Был предложен алгоритм совместного определения частот, в его основу легла функция набора кандидатов. После получения набора вероятно участвующих нот, программой определяются гармоники, которые были наложены друг на друга, и они сглаживаются. Оптимальный набор выбирается на основе оценки максимального правдоподобия.

2.5. Статистический подход

Многие из описанных подходов для решения проблемы мультичастотного определения руководствуются математической статистикой. Проблема может быть сведена к проблеме апостериорного максимума, если на каждом исследуемом отрезке v будет взят набор возможных комбинаций фундаментальных частот [3].

$$\hat{C} = \arg \max_{C \in \mathbb{C}} P(C | v) \quad (2.4)$$

В отсутствие дополнительной информации задачу можно решать по правилу Байеса как проблему максимального правдоподобия.

Подкрепляя алгоритм знанием теории музыки, можно расширить применение теории вероятностей. Например, в попытке определить ноту, программа в первую очередь будет рассматривать ноты, которые попадают в тональность, а не диссонируют с соседними. С другой стороны, такие предположения накладывают ограничения на музыкальный стиль, исследуемого произведения.

2.6. Методы разложения спектрограммы

Многие подходы к транскрибированию музыки используют техники факторизации спектрограммы. К таким техникам относятся разложение неотрицательных матриц и probabilistic latent component analysis – его вероятностный аналог.

Разложение матриц является инструментом, раскладывающим входящую спектрограмму.

Матрицы считаются вероятностными распределениями в латентно-компонентном анализе. Это утверждение позволяет, избегая трудностей, формализовать модель.

2.7. Методы с обучением, полуавтоматический подход

Для оценки фундаментальной частоты в некоторых методах используются техники стандартного машинного обучения. Так для распознавания нот были натренированы нейронные сети, они способны отслеживать гармоники каждой частоты. В множестве систем для запуска процесса распознавания от пользователя требуется ввести предварительную информацию. Эти системы нецелесообразно применять к большим базам музыки, но они могут быть

полезны в масштабах потребностей одного человека – музыканта или музыковеда. Для полуавтоматического подхода характерно перекладывание части задач на человека. Тем не менее, такого рода методы не должны быть слишком затратными по времени и выдавать точную информацию, которой являются, например, тональность, информация об инструментах, темп и размер.

При решении задачи разделения источников системы, алгоритмы, взаимодействующие с пользователем для получения предварительной интонации, показали лучшие результаты, нежели «слепые» методы.

2.8. Применение генетических алгоритмов

Генетические алгоритмы — это подмножество эволюционных алгоритмов, использующих такие техники, вдохновлённые эволюционной биологией, как наследование, мутация, селекция и рекомбинация. Генетический алгоритм — это метод, используемый для приближённого решения задач оптимизации. Генетические алгоритмы также считают эвристиками глобального поиска.

Генетические алгоритмы реализуются как компьютерная симуляция [1], в которой есть генотипы и фенотипы. Фенотипы (потенциальные решения) являются носителями различных генотипов (абстрактных представлений, связанных с задачей). В ходе симуляции вырабатываются новые фенотипы (более оптимальные решения задачи). Обычно решения представляются бинарными строками (строками из нулей и единиц), однако бывают и другие варианты. Эволюция начинается со случайно сгенерированной популяции и затем происходит поэтапно - поколение за поколением. В каждом поколении выбираются оптимальные фенотипы, затем они изменяются (с помощью рекомбинации) для формирования нового поколения. Алгоритм завершается после определённого количества поколений или когда достигнут желаемый уровень оптимальности хотя бы у одного фенотипа.

Генетические алгоритмы находят применение в биогенетике, компьютерных науках, инженерии, экономике, химии и других областях.

Ниже представлены некоторые соображения по поводу применения генетических алгоритмов к задаче транскрипции музыки.

Каждый генотип будет соответствовать допустимому решению и будет состоять из последовательности нот (количество нот тоже скорее всего будет различаться). Нота здесь представлена не только высотой звука, но и её громкостью, длительностью и начальным временем. Также иногда полезно добавлять к информации ноты сведения о соответствующем инструменте.

Начальная популяция будет способствовать на получения лучшего результата по двум "пунктам":

1. Если первое поколение будет ближе к идеальному результату, чем сгенерированное случайным образом, то для нахождения результата понадобится существенно меньшее количество поколений.
2. Первое поколение должно быть достаточно неоднородным чтобы покрыть максимальное количество возможных способов решения задачи.

Первый шаг к получению наиболее адекватного результата для первого поколения заключается в анализе входного аудиопотока, Например создание фенотипов, чьи ноты будут подбираться на основании спектрограммы потока. Конечно, в таком случае будет достаточно много ошибок на целое число октав, но это не страшно, так как это лишь первое поколение.

Выбор "отцовских" генотипов осуществляется типичным для генетических алгоритмов образом (по оптимальности), то же можно сказать о рекомбинации (создаём новые фенотипы в зависимости от отцовских). Стоит также помнить о том, что нам нужны генотипы различных размеров. Чтобы применять мутации, алгоритм должен быть способен к удалению нот, их добавлению и изменению (то есть к изменению отдельных параметров ноты - высоты, громкости и т. п.).

Функция оптимальности сильнее всего влияет на результат работы генетического алгоритма. Кажется очевидным, что мы должны рассматривать "похожесть" фенотипа и оригинального аудиопотока как функционал качества в данной задаче. Первая проблема заключается в том, что мы должны уметь синтезировать аудиопоток по генотипу (для возможности вычисления значения

функционала). Второй вопрос заключается в том, как именно сравнивать синтезированный поток с оригинальным.

Для решения вышеупомянутых проблем применяют 4 основных подхода:

1. Использовать для всех нот фенотипов один инструмент;
2. Использовать различные инструменты, но тогда придётся наделять каждую ноту информацией об инструменте;
3. Использовать отдельный генетический алгоритм для построения подходящего синтезатора. Цель в получении синтезатора, дающего звуки наиболее похожие на звуки из оригинального аудиопотока;
4. Использование второго и третьего подхода одновременно: использование нескольких синтезаторов (для моделирования нескольких инструментов) и указание для каждой ноты информации о том, каким синтезатором она должна быть исполнена.

Тем не менее, важно помнить, что даже если звуки, созданные синтезатором, недостаточно похожи на звуки оригинальной аудиодорожки, правильный выбор функционала качества позволяет это обойти. В конце концов нам нужен наиболее похожий результат, а не точно такой же.

Просто сравнивать аудиопотоки как две функции времени бессмысленно, так как инверсия фазы уже может привести к тому, что мелодии будут считаться абсолютно разными. Например, если использовать разницу амплитуд результатов преобразования Фурье на коротких временных промежутках, то результат может быть и не таким точным, зато точно будет более реалистичным.

2.9. Выбор метода

Алгоритм вейвлет-преобразования обладает высокой эффективностью в области обработки звуковых сигналов, кроме того, он устойчив к воздействию помех. Это делает его более перспективным, чем традиционно используемый алгоритм Фурье.

В вейвлет-преобразовании мы можем сформировать семейство вейвлетов $w_{s,\tau}(t)$ с помощью сдвигов τ и масштабирования s материнского вейвлета $w(t)$. Формирование семейства вейвлетов схоже с системой формирования семейств нот $n_i^k(t)$ из одной базовой $n_0^k(t)$ путем сдвигов θ_i и масштабирования m_i . По своему принципу непрерывное вейвлет-преобразование очень подходит данной задаче.

ГЛАВА 3. НЕПРЕРЫВНОЕ ВЕЙВЛЕТ- ПРЕОБРАЗОВАНИЕ В ЗАДАЧЕ ТРАНСКРИПЦИИ СИГНАЛА

Непрерывное вейвлет-преобразование выглядит следующим образом:

$$Wf(\tau, s) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{s}} f(t) w\left(\frac{t-\tau}{s}\right) dt, \quad (3.1)$$

где $w(t)$ – материнский вейвлет;

s – коэффициент масштабирования вейвлета;

τ – коэффициент сдвига вейвлета.

Для каждого масштаба s функция Wf_s равна взаимной корреляции дочернего вейвлета $w_s(t)$ и исходного сигнала $f(t)$, что описывает степень их схожести:

$$Wf_s(\tau) = \int_{-\infty}^{\infty} f(t) w_s(t - \tau) dt \quad (3.2)$$

$$w_s(t) = w\left(\frac{t}{s}\right) \quad (3.3)$$

Идея непрерывного вейвлет-преобразования основана на формировании базисной функции определенного вида, называемой материнским вейвлетом. Материнский вейвлет $w(t)$ обладает свойством масштабируемости, что позволяет получить семейство вейвлетов, настроенных на определенный спектральный диапазон.

Исследования свойств вейвлет-функций показали, что наилучшие графические представления результатов НВП получены при схожести частотных спектров сигнала $f(t)$ и вейвлета $w(t)$ [5]. Предполагается, что повышение информативности результатов возможно при абсолютном совпадении вейвлет-функции и исследуемого сигнала в каждый момент времени.

В качестве базовой ноты n_0 может быть выбрана любая нота из диапазона определенного инструмента, вне зависимости от использования в конкретном

музыкальном сигнале. Например, за $n_0(t)$ можно принять временную функцию ноты «ля» первой октавы, с частотой основного тона $\nu = 440$ Гц [4].

Большинство музыкальных инструментов обладают свойством автомодельности [7]. Это свойство позволяет из временной функции одной ноты $n_0(t)$ определенного музыкального инструмента получить временную функцию любой другой ноты $n_i(t)$ этого музыкального инструмента масштабированием функции $n_0(t)$ вдоль оси времени:

$$n_i(t) = n_0\left(\frac{t}{m_i}\right), \quad (3.4)$$

где m_i — коэффициент масштабирования;

i — положение ноты $n_i(t)$ по высоте относительно ноты $n_0(t)$.

Для равномерно темперированного строя европейской музыки $m_i = 2^{-i/12}$, где $i \in \mathbb{Z}$. Если рассматривать ноту "до" первой октавы как базовую (n_0), то характеристика ноты "до" следующей октавы может быть представлена как $n_{12}(t) = n_0\left(\frac{t}{2^{12/12}}\right)$.

Предлагается сформировать собственный материнский вейвлет (а не пользоваться стандартными). Его можно сформировать из локального участка функции сигнала базовой ноты этого инструмента $n_0(t)$ (рис. 3.1):

$$w(t) = n_0(t + t_0), \text{ при } t \in [0, T] \quad (3.5)$$

$$w(t) = 0, \text{ при } t \notin [0, T] \quad (3.6)$$

$$n_0(t_0) = 0 \quad (3.7)$$

$$n_0(t_0 + T) = 0 \quad (3.8)$$

Для задач выявления элементарных составляющих (объектов нот) в музыкальном сигнале, необходимо формирование семейства вейвлетов удовлетворяющих двум условиям:

- разрешение по частоте (ширина окна вдоль оси масштабов Δs) должно обеспечивать таким, чтобы частоты основных тонов двух соседних ноты были различимы;

- разрешение по времени (ширина окна вдоль оси времени $\Delta\tau$) должно позволять выполнять идентификацию всех нот минимально возможной длительности [6].

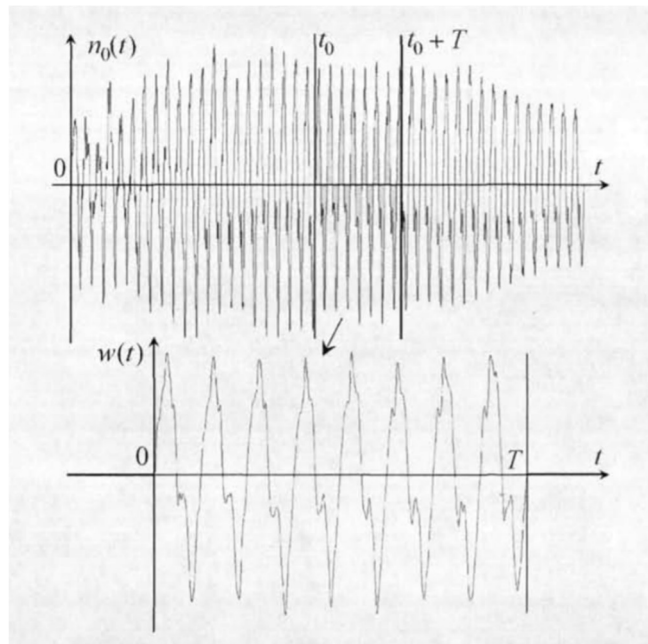


Рис. 3.1 – Функция $n_0(t)$ и сформированный на основе ее фрагмента вейвлет $w(t)$

Свойство автомодельности сигналов музыкальных инструментов позволяет сформировать и применять только один вейвлет $w(t)$ для идентификации любой ноты одного музыкального инструмента.

НВП сигнала $f(t)$ состоит в его разложении по некоторому базису, сконструированному из функции-вейвлета $w(t)$, посредством ее масштабирования и сдвигов вдоль оси времени:

$$[W_w f](\tau, s) = |s|^{-\frac{1}{2}} \int_{-\infty}^{\infty} f(t) w^* \left(\frac{t - \tau}{s} \right) dt, \quad (3.9)$$

где $[W_w f](\tau, s)$ — коэффициенты преобразования;

s — масштаб вейвлета;

τ — параметр смещения вдоль оси времени;

«*» — операция комплексного сопряжения.

Вычисление НВП, представленного в аналитическом виде, при цифровой реализации имеет следующие ограничения:

- частота дискретизации ν музыкального сигнала $f(t)$ и базисного вейвлета $w(t)$ равняется 44100 с^{-1} ;
- значение шага сдвига вейвлета $\Delta\tau = 44100^{-1} \text{ с}$;
- коэффициент масштабирования вейвлета $s_i = 2^{\frac{i}{12}}$;
- количество частотных уровней S в выходном массиве $Wf(\tau, s) = 96$.

Результатом НВП является двумерный массив чисел $Wf(\tau, s)$, характеризующий амплитудно-частотно-временные свойства сигнала. Значение числа в каждом элементе массива соответствует амплитуде определенной гармонике исследуемого сигнала в различные моменты времени.

Выделяется два этапа в решении задачи классификации музыкальных объектов:

1. классификация амплитудно-частотных составляющих объектов (определение тона m образов нот музыкального инструмента определенного тембра n_0 в каждый момент времени);
2. классификация временных характеристик объектов (определение времени начала θ и длительности Θ образа O_i).

ГЛАВА 4. ПРИМЕНЕНИЕ НЕПРЕРЫВНОГО ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЯ

Применение непрерывного-вейвлет преобразования (НВП) в задаче транскрипции музыкального файла можно разделить на следующие этапы:

1. Разработка алгоритма непрерывного вейвлет-преобразования;
2. Выбор материнского вейвлета;
3. Создание семейства вейвлетов на основе материнского, или разработка алгоритма масштабирования материнского вейвлета;
4. Применение алгоритма непрерывного вейвлет-преобразования;
5. Обработка данных (фильтрация);
6. Нахождение нотных объектов в обработанных данных и создание транскрипции.

Алгоритм НВП представляет из себя (Приложение 1):

1. Получаем на вход исходный сигнал и список масштабов, или нот, существование которых нужно проверить в сигнале;
2. Для каждого масштаба s создаем подходящий дочерний вейвлет w_s длиной в 16 периодов;
3. Рассчитываем Wf_s – результат взаимной корреляции между исходным сигналом и w_s ;
4. Умножаем Wf_s на $|modifier|^{-1/2}$ согласно формуле (3.9);

Рассмотрим применение непрерывного-вейвлет преобразования (НВП) в задаче транскрипции синусоидального сигнала и сигнала, приближенного к реальному.

4.1. Синусоидальный сигнал

4.1.1. Получение сигнала

Для начала испытаем алгоритм на синусоидальном сигнале - трезвучие «до-мажор» первой октавы (Приложение 2). Частоты основных тонов нот этого

трезвучия соответствуют гармоническим сигналам с частотами 261,6, 329,6 и 392 Гц. Продолжительность сигнала выбрана равной 0,2 с:

$$f(t) = \sin(2\pi 261,6t) + \sin(2\pi 329,6t) + \sin(2\pi 392t) \quad (4.1)$$

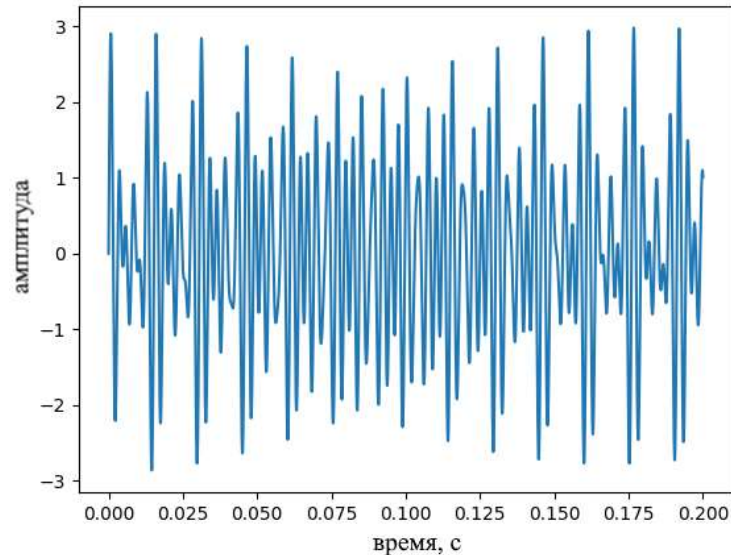


Рис. 4.1 – График тестового сигнала

4.1.2. Создание вейвлета

Определим материнский вейвлет как синусоиду с частотой $n_0 = 440$ Гц:

$$w_0(t) = \sin(2\pi n_0 t) \quad (4.2)$$

Чтобы использовать функцию как вейвлет, ее интеграл должен равняться нулю. Так как функция четная и мы используем целое число периодов, условие соблюдается.

Тогда дочерний вейвлет для масштаба s примет вид:

$$w_s(t) = \sin\left(2\pi * n_0 * \frac{t}{modifier_s}\right) \quad (4.3)$$

$$modifier_s = 2^{-s/12}, s \in \mathbb{Z} \quad (4.4)$$

Лучше всего НВП себя покажет, если в дочернем вейвлете будет 16 периодов [6].

4.1.3. Применение НВП

Проведем НВП с данным вейвлетом при масштабе от -36 до 36, т. е. в пределах шести октав. Результат НВП представлен на рис. 4.2.

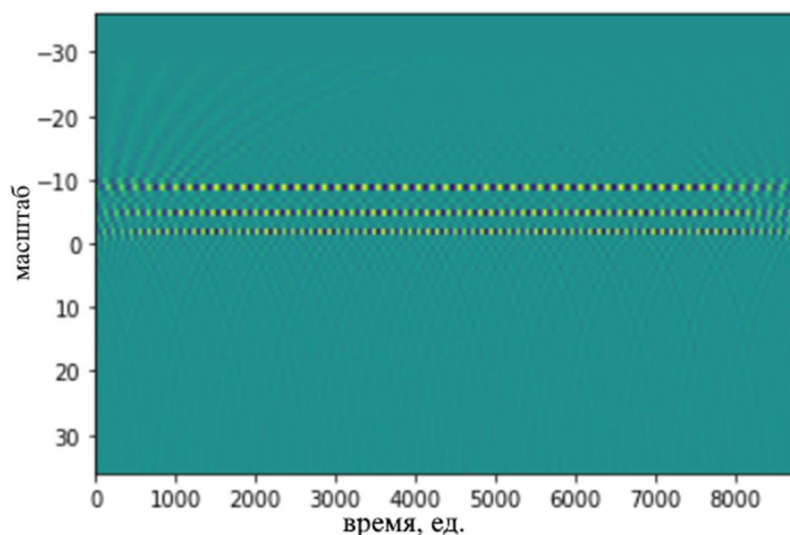


Рис. 4.2 – Скалограмма НВП без фильтров

Мы получили двумерную матрицу, где каждая строчка представляет собой результат взаимной корреляции исходного сигнала и вейвлета указанного масштаба.

4.1.4. Обработка данных

Обработаем результат НВП. Применим следующие фильтры (рис. 4.3):

1. Возьмем абсолютные значения;
2. `maximum_filter1d` модуля SciPy;
3. `median_filter1d` модуля SciPy;
4. Фильтр порога по максимуму на строке (Приложение 3).

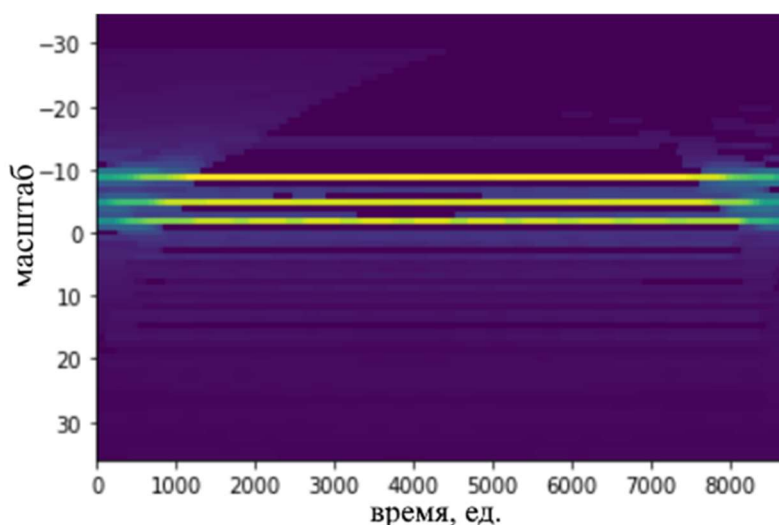


Рис. 4.3 – Скалограмма НВП с фильтрами по строке

Далее применим следующие фильтры на столбцах (рис. 4.4):

1. Фильтр порога по максимуму на столбце (Приложение 3).
2. Фильтр локального максимума по столбцу: если элемент больше, чем два предыдущих и два следующих, помечаем единицей, иначе – нулем (Приложение 3).

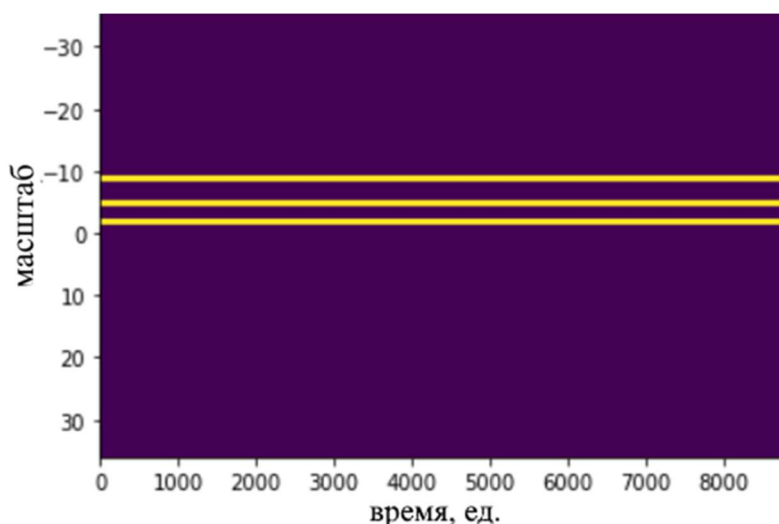


Рис. 4.4 – Скалограмма НВП с фильтрами по строке и столбцу

4.1.5. Поиск нотных объектов

Для поиска нотных объектов в полученных данных воспользуемся следующим алгоритмом (Приложение 4):

1. На строке найдем индексы начала цепочек повторяющихся сигналов «нота присутствует» и «нота отсутствует».
2. Получим длины цепочек повторений.
3. Отбросим все цепочки отсутствующих нот, чьи длины меньше установленной минимальной длины. Таким образом можно заполнить незначительные разрывы в нотах – сгладить данные.
4. Если длина цепочки присутствующей ноты больше установленной минимальной длины, записываем ноту. Ее начало будет в начале цепочки повторяющихся сигналов «нота присутствует». Длина ноты – длина цепочки единиц. Частоту ноты можно рассчитать с помощью масштаба s :

$$Fc = \frac{Fc_0}{2^{-s/12}}, \quad (4.6)$$

где Fc – частота ноты;

Fc_0 – частота базовой ноты;

s – масштаб.

4.1.6. Пример работы программы

Испытаем алгоритм на более сложном сигнале. Составим сигнал на основе данных, представленных в табл. 4.1.

Таблица 4.1

Исходные данные синусоидального сигнала

Частота, Гц	Исходные данные	
	Начало ноты, с	Длина ноты, с
261,63	0	0,2
329,63	0	0,2
369,99	0,4	0,04
392,00	0	0,2
523,25	0	0,2
	0,4	0,04
622,25	0,2	0,4

Уравнение сигнала:

$$\left\{ \begin{array}{l} f_1(t_1) = \sin(2\pi 261,63 t_1) + \sin(2\pi 329,63 t_1) + \\ \quad \sin(2\pi 392 t_1) + \sin(2\pi 523,25 t_1) \\ f_2(t_2) = \sin(2\pi 622,25 t_2) \\ f_3(t_3) = \sin(2\pi 369,99 t_3) + \sin(2\pi 523,25 t_3) + \sin(2\pi 622,25 t_3) \\ f_4(t_4) = \sin(2\pi 622,25 t_4) \\ f(t) = f_1 + f_2 + f_3 + f_4 \\ t_1 \in [0; 0,2], t_2 \in (0,2; 0,4], t_3 \in (0,4; 0,44], t_4 \in (0,44; 0,6], t \in [0; 0,6] \end{array} \right. \quad (4.5)$$

Построим график сигнала (рис. 4.5). Применим НВП и фильтры. Результаты представлены на рис. 4.6.

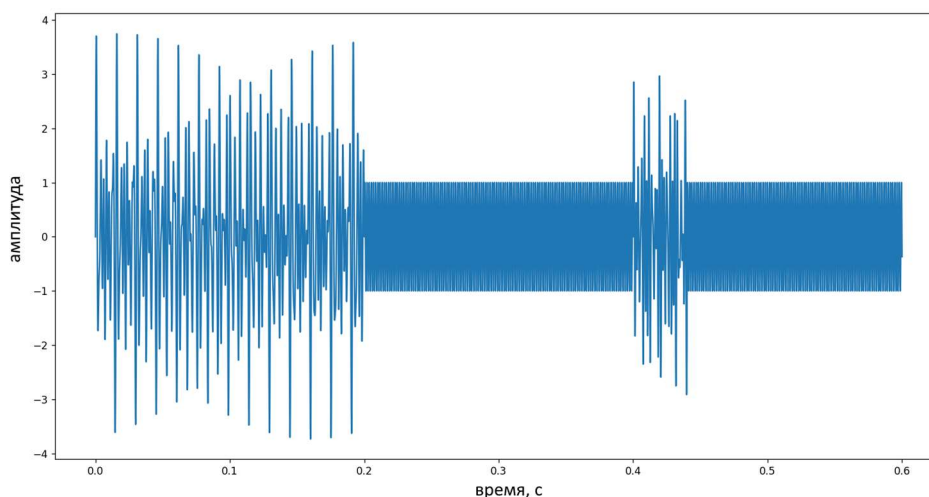


Рис. 4.5 – График синусоидального сигнала

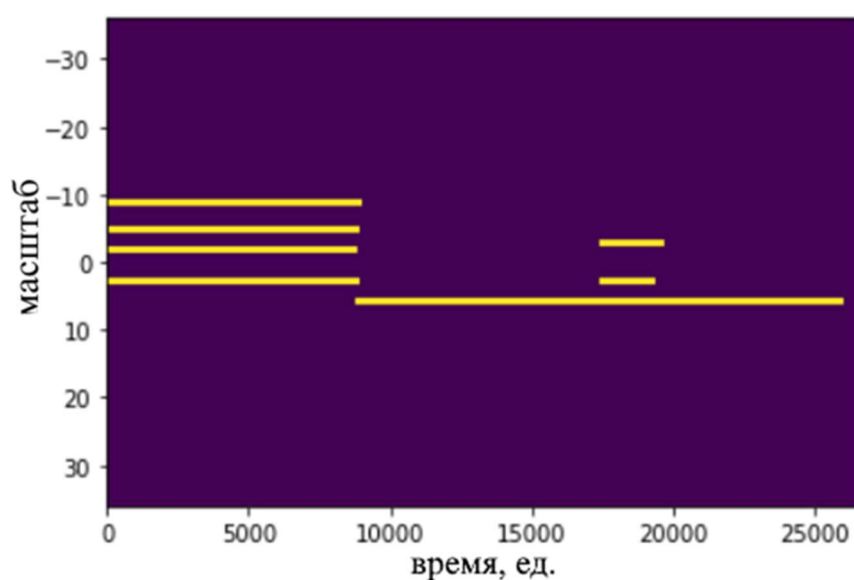


Рис. 4.6 – Скалограмма НВП с фильтрами

На рис. 4.6 четко видны прямые линии, масштаб точно соответствует используемым частотам. Переведем масштаб в частоту, узнаем начало и длину каждой ноты. Сравнение с исходными данными представлено в табл. 4.2.

Таблица 4.2

Сравнение исходных данных и результатов программы

Частота, Гц	Исходные данные		Результат программы		Разница	
	Начало ноты	Длина ноты	Начало ноты	Длина ноты	Начало ноты	Длина ноты
261,63	0	0,2	0	0,2	0	0

Продолжение табл.4.2

329,63	0	0,2	0	0,2	0	0
369,99	0,4	0,04	0,39	0,05	-0,1	0,1
392,00	0	0,2	0	0,2	0	0
523,25	0	0,2	0	0,2	0	0
	0,4	0,04	0,39	0,05	-0,1	0,1
622,25	0,2	0,4	0,2	0,39	0	-0,1

С незначительной погрешностью результат соответствует исходным данным. 0,04 секунды – время, соответствующее 1/32 ноты, самой короткой ноте, с которой будем работать. Погрешность начала и длины нот не превышает этого числа. Можно считать эксперимент успешным и перейти к анализу сигнала, приближенного к настоящему.

4.2. Сигнал, приближенный к реальному

4.2.1. Получение сигнала

Получим сигнал с помощью синтезатора рояля в FL Studio. Сыграем октаву, снимок из программы представлен на рис. 4.7. График сигнала музыкального файла представлен на рис. 4.8. Код программы представлен в Приложении 5.

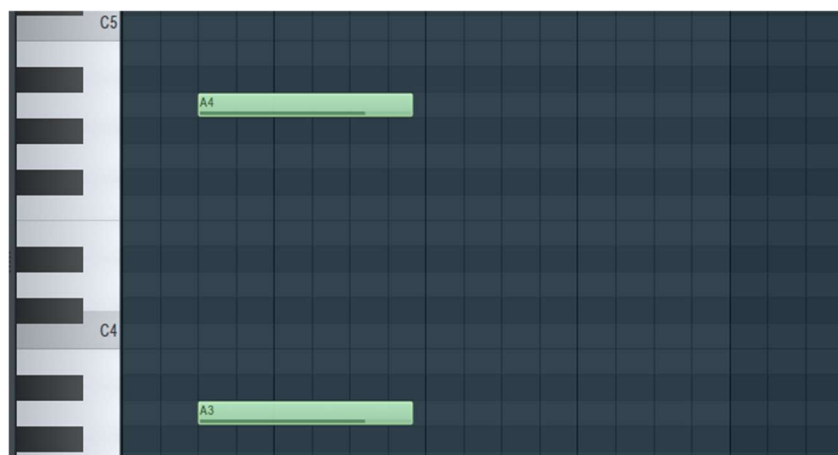


Рис. 4.7 – Октава на рояле в FL Studio

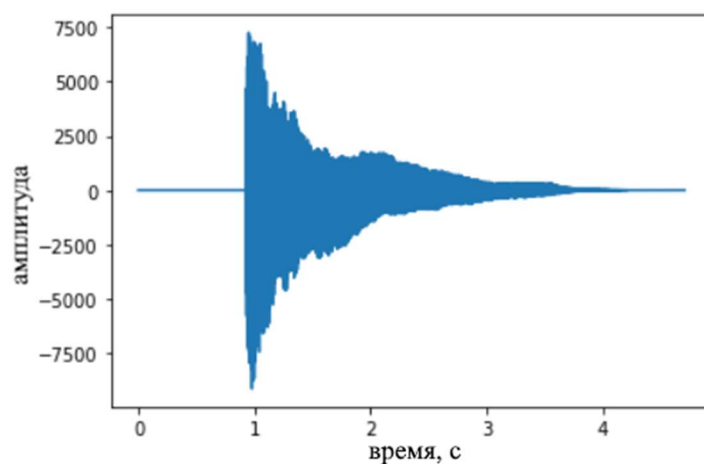


Рис. 4.8 – График исходного сигнала (октава A3 и A4)

4.2.2. Создание вейвлета

Для создания вейвлета сыграем в синтезаторе рояля несколько раз базовую ноту – ноту «ля» первой октавы, или A4 (рис. 4.9).

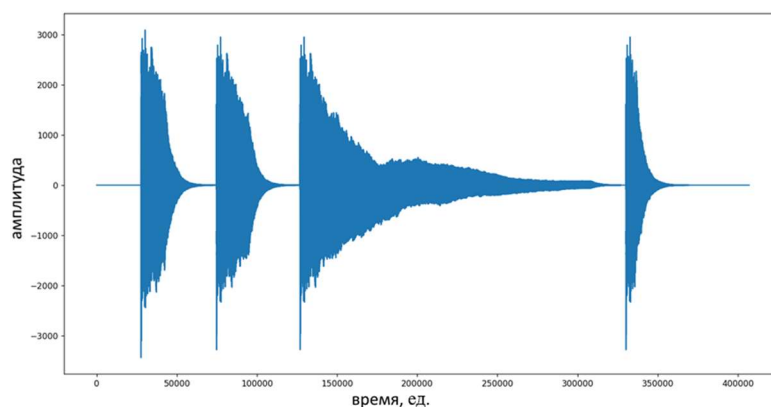


Рис. 4.9 – График сигнала четырех нот A4

Возьмем ее фрагмент длиной примерно в 16 периодов (рис. 4.10).

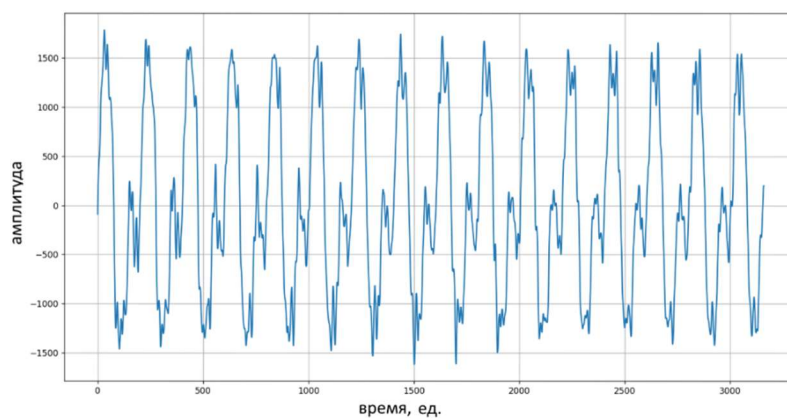


Рисунок 4.10 – График фрагмента сигнала ноты A4

Интеграл фрагмента будет равен:

$$\int_{-\infty}^{\infty} f(x) = \int_0^{size} f(x) = \sum_{n=0}^{n=size} f(n) = 9932, \quad (4.6)$$

где *size* – длина фрагмента.

Чтобы интеграл равнялся нулю, обработаем фрагмент:

1. Применим прямое преобразование Фурье для дискретного сигнала.
2. Получив массив значений, первое значение обнулим.
3. Применим обратное преобразование Фурье для дискретного сигнала на измененном массиве.
4. Посчитаем сумму элементов. Она приблизительно равна нулю ($-3 * 10^{-14}$), что подходит для вейвлета.

График обработанного фрагмента сигнала представлен на рис. 4.11.

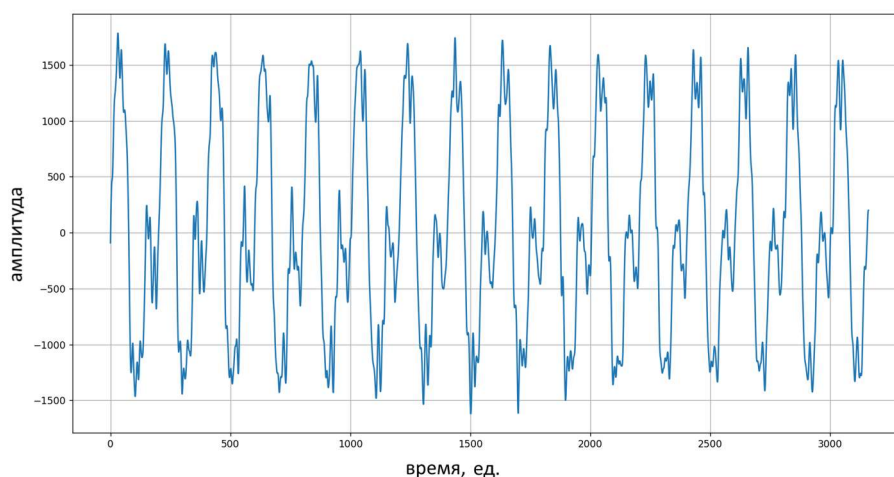


Рис. 4.11 – График обработанного фрагмента ноты А4, его интеграл равен нулю

Таким образом мы получили материнский вейвлет. Дочерний вейвлет можно получить, масштабируя материнский до нужного размера с помощью кусочно-линейной аппроксимации.

Для всех допустимых нот $N_1 \dots N_j$ определим:

$$modifier_s = 2^{-s/12}, s \in \mathbb{Z} \quad (4.7)$$

$$daughter_{длина} = mother_{длина} * modifier_s \quad (4.8)$$

$$x_d = 0,1, \dots, daughter_{\text{длина}} \quad (4.9)$$

$$x_m = \frac{x_d}{modifier_s} \quad (4.10)$$

$$\begin{aligned} daughter(x_d) = mother(\lfloor x_m \rfloor) + \\ + (mother(\lfloor x_m + 1 \rfloor) - mother(\lfloor x_m \rfloor)) * \\ * (x_m - \lfloor x_m \rfloor) \end{aligned} \quad (4.11)$$

где s — положение ноты $N_i(t)$ по высоте относительно ноты $N_0(t)$;

$modifier$ — коэффициент масштабирования;

$daughter$ — дочерний вейвлет;

$mother$ — материнский вейвлет;

x_d — координата дочернего вейвлета;

x_m — координата материнского вейвлета, соответствующая x_d ;

$\lfloor x_m \rfloor$ — целая часть x_m .

4.2.3. Применение НВП

Проведем НВП с данным вейвлетом при масштабе от -24 до 12., т. е. в пределах трех октав. Скалограмма результатов представлена на рис. 4.12.

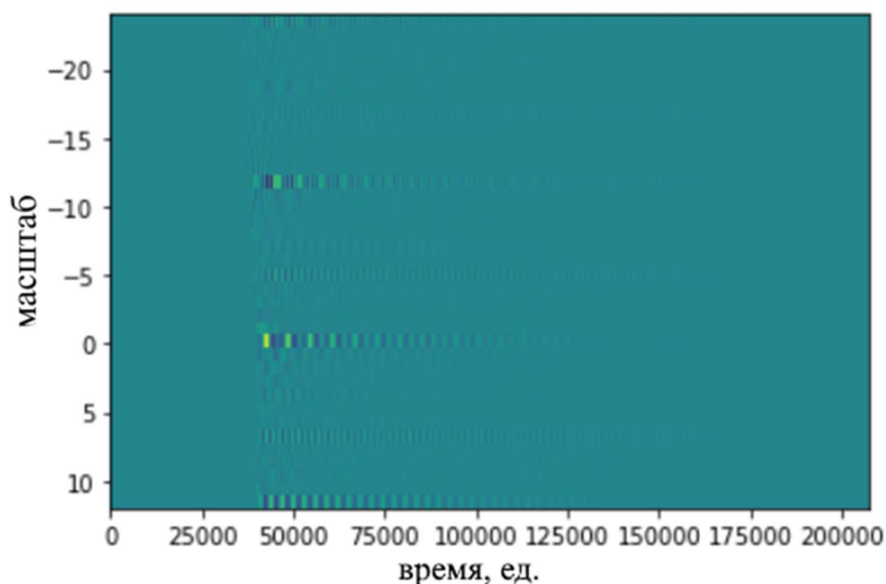


Рис. 4.12 – Скалограмма НВП без фильтров

4.2.4. Обработка данных

Обработаем результат НВП. Применим следующие фильтры (рис. 4.13):

1. Возьмем абсолютные значения;
2. `maximum_filter1d` модуля SciPy;
3. `median_filter1d` модуля SciPy;

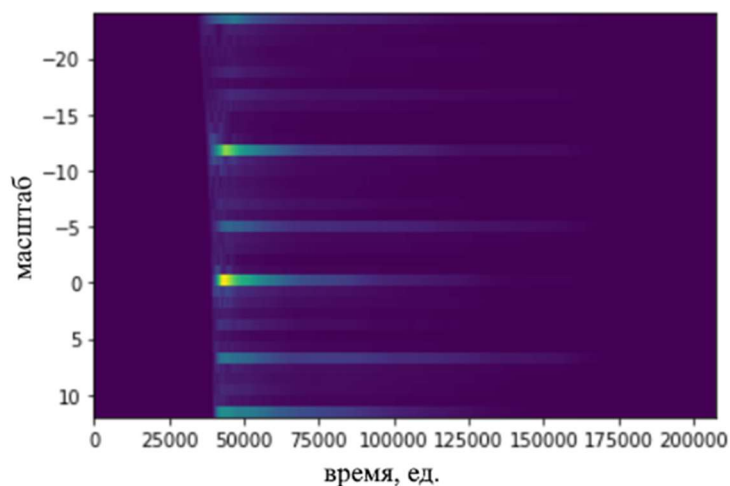


Рис. 4.13 – Скалограмма НВП с фильтрами по строке

Далее применим следующие фильтры на столбцах (рис. 4.14):

1. Вертикально-октавный фильтр, чтобы избавиться от гармонических ошибок (ложных октав) и оставить только действительно прозвучавшие (Приложение 3);
2. Фильтр порога по максимуму на столбце.

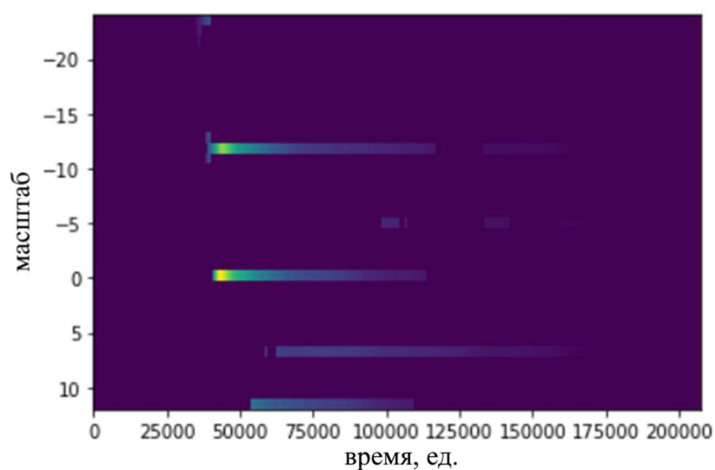


Рисунок 4.14 – Скалограмма НВП с фильтрами по строке и столбцу

4.2.5. Нахождение нотных объектов

Найдем максимальное значение во всей матрице. Пусть пороговое значение для существования потенциальной ноты равняется некоторой доли от этого значения.

Найдем максимальное значение на каждой строке. Если оно больше, чем пороговое значение для существования потенциальной ноты, то на этой строке есть потенциальная нота.

На все строки с потенциальными нотами применим следующие фильтры:

1. `maximum_filter1d` модуля SciPy;
2. `median_filter1d` модуля SciPy;

Рассмотрим отклик непрерывного вейвлет-преобразования во время звучания ноты (рис. 4.15).

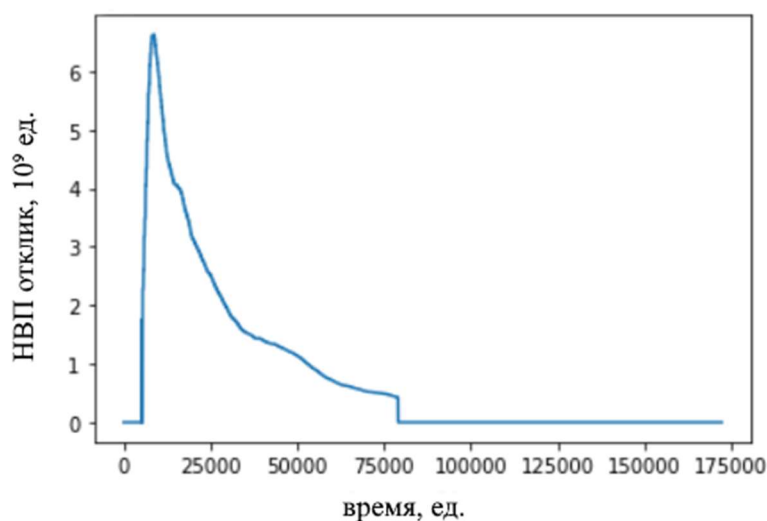


Рис. 4.15 – Пример зависимости отклика НВП от времени при звучании ноты

Можно заметить, что при нажатии на ноту отклик значительно повышается, затем постепенно уменьшается. Чтобы найти моменты нажатия на ноты, нужно следить за резким повышением отклика НВП.

В каждой строке с потенциальной нотой ищем начала нот следующим образом:

1. Возьмем модуль производной строки. Это даст нам разницу между соседними элементами.
2. Применим фильтр порога по максимуму (Приложение 3): должны остаться только резкие изменения значений.
3. Находим места заметных пиков и длины между ними (рис. 4.16).
4. Фильтруем по длине до ближайшего пика: оставляем только значения больше минимальной установленной длины. Это необходимо, чтобы найти начало пика, отфильтровав все значения, которые принадлежат этому же пику. В результате получаем места начала потенциальных нот. Например, на рисунке 4.16 их пять.
5. Имея начала потенциальных нот на строке, можно разделить строку на промежутки: от начала одной потенциальной ноты до начала следующей (или до конца строки, если следующей ноты нет).
6. Найдем максимальный отклик в промежутке и объявим его местоположение началом ноты. Наблюдаем за уменьшением отклика НВП после этого: как только оно достигнет минимального установленного значения, объявляем это место концом ноты (рис. 4.17).
7. Если длина получившейся ноты больше установленной минимальной длины ноты, записываем ноту.

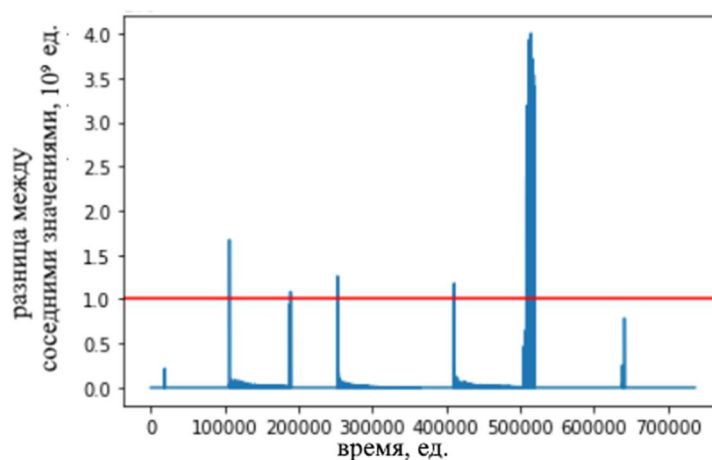


Рис. 4.16 – Пример работы поиска пиков

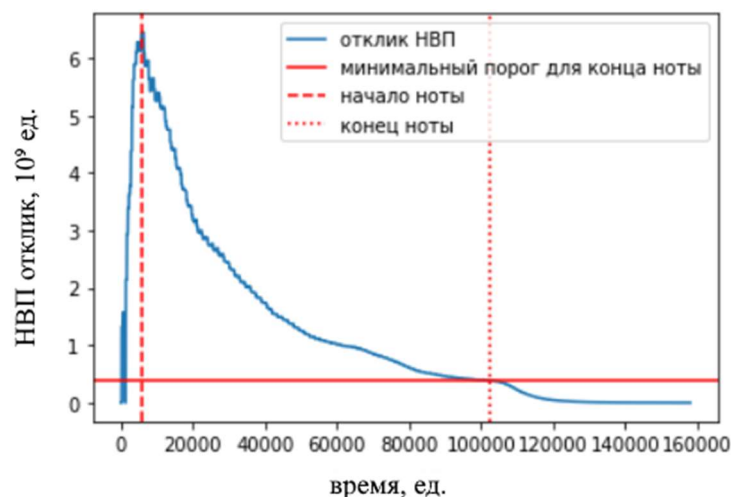


Рис. 4.17 – Пример определения начала и конца ноты на интервале

Код программы представлен в Приложении 6.

Отобразим записанные ноты на скалограмме (рис. 4.18).

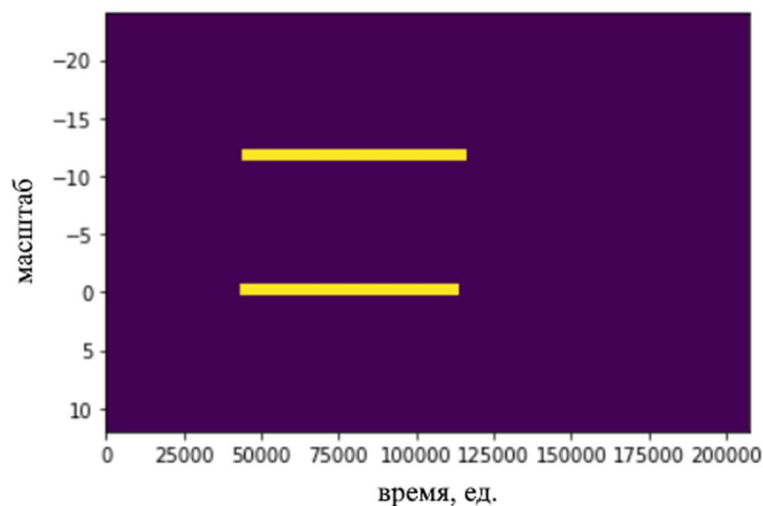


Рис. 4.18 – Скалограмма с найденными нотами

Сохраним записанные ноты как MIDI файл (Приложение 7).

4.2.6. Об использовании подходящего вейвлета

Протестируем работу алгоритма при использовании разных материнских вейвлетов. Первый вейвлет будет принадлежать тому же источнику, что и исходный сигнал (то есть получен с помощью синтезатора рояля FL Studio), второй будет принадлежать другому источнику – синтезатору рояля OnlinePiano.

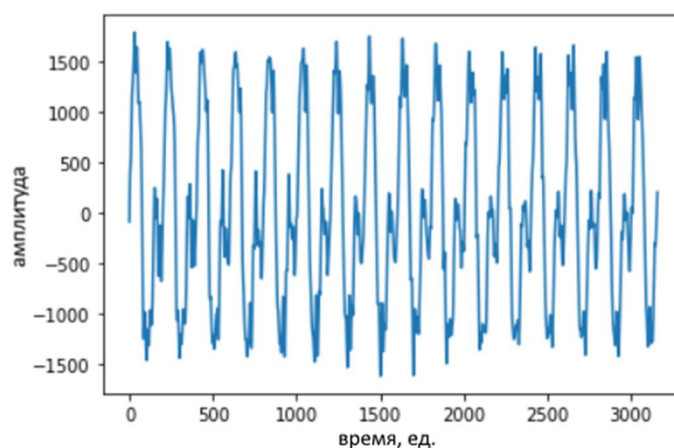


Рис. 4.19 – Материнский вейвлет источника сигнала

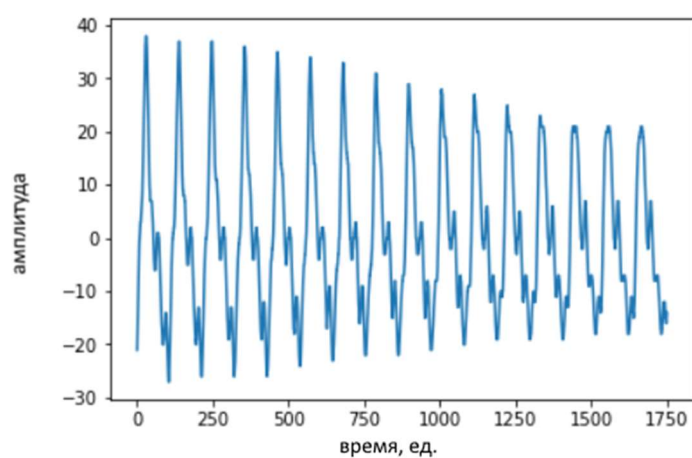


Рисунок 4.20 – Материнский вейвлет другого источника

В качестве примера возьмем аккорд октавы, полученный с помощью FL Studio. Применим НВП, возьмем абсолютные значения и применим фильтр максимума по строке и медианный фильтр по строке.

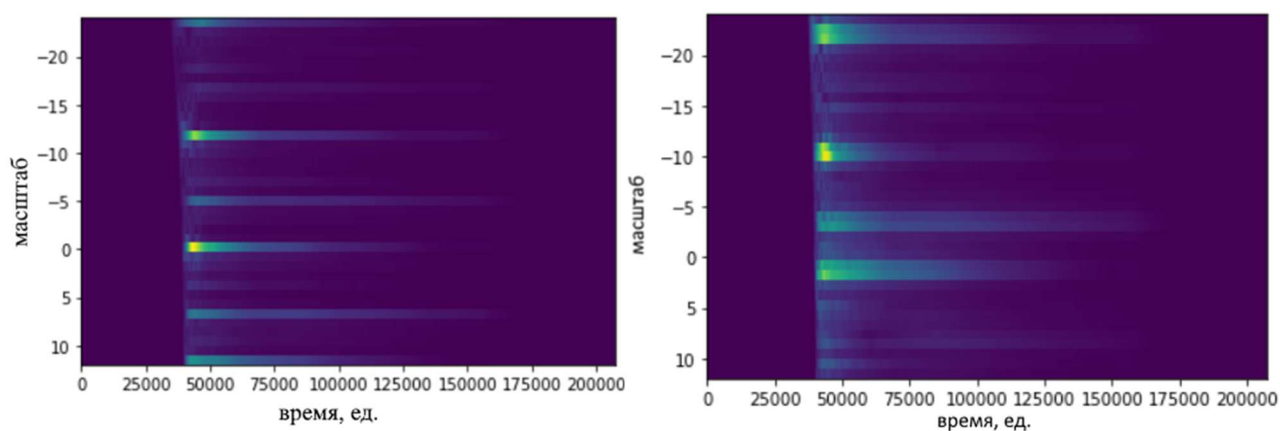


Рис. 4.21 – Сравнение результатов НВП с подходящим вейвлетом (слева) и неподходящим вейвлетом (справа)

Алгоритм НВП действительно дает более точные результаты, если использовать подходящий источнику сигнала вейвлет. Максимальный отклик НВП с подходящим вейвлетом превысил отклик НВП с неподходящим в 163 раза.

Здесь рассмотрено сравнение вейвлетов, принадлежащих одному инструменту – роялю. Однако по результату можно с уверенностью предположить: если использовать вейвлеты и других инструментов (скрипка, орган и т. д.), можно будет различать играющие одновременно инструменты и получать для них их собственную партитуру.

ГЛАВА 5. ОЦЕНКА РАБОТЫ ТРАНСКРИПЦИИ

5.1. Характеристики оценки

Эффективность системы музыкальной транскрипции рассчитывается с использованием различных параметров.

В работе введены ряд характеристик оценки качества распознавания, на основе предлагаемых авторами систем идентификации и классификации объектов музыкальных сигналов [8]:

1. Относительное количество нот, совпавших в распознаваемом и оригинальном произведениях:

$$\Delta K_c = \frac{K_c}{K_o} \cdot 100\%, \quad (5.1)$$

где K_c — общее количество совпавших нот;

K_o — общее количество нот в произведении.

Ноты считаются совпавшими, если имеют пересечение во времени, то есть до момента затухания одной ноты, начала звучать другая.

2. Среднее относительное отклонение времени начала звучания каждой ноты:

$$\Delta \theta = \frac{\sum_i \frac{\Delta \theta_i}{\Theta_i}}{K_o} \cdot 100\%, \quad (5.2)$$

где $\Delta \theta_i$ — разность во времени начала звучания ноты i образа и объекта;

Θ_i — продолжительность звучания ноты i .

3. Среднее относительное отклонение времени продолжительности звучания каждой ноты:

$$\Delta \Theta = \frac{\sum_i \frac{\Delta \Theta_t}{\Theta_t}}{K_o} \cdot 100\%, \quad (5.3)$$

где $\Delta \Theta_t$ — разность в продолжительности звучания ноты i образа и объекта.

4. Относительное количество распознанных нот:

$$\Delta K_P = \frac{K_P}{K_O} \cdot 100\%, \quad (5.4)$$

где K_P — количество нот, идентифицированных системой.

5. Относительное количество нераспознанных нот:

$$\Delta K_H = \frac{K_H}{K_O} \cdot 100\%, \quad (5.5)$$

где K_H — количество нот, присутствовавших в сигнале-источнике и не распознанных системой.

5.2. Анализ работы алгоритма

Эксперимент 1. Проверим работу алгоритма на октаве. Визуализация исходного и полученного MIDI-файлов представлена на рис. 5.1. Оценка качества распознавания представлена в табл. 5.1.

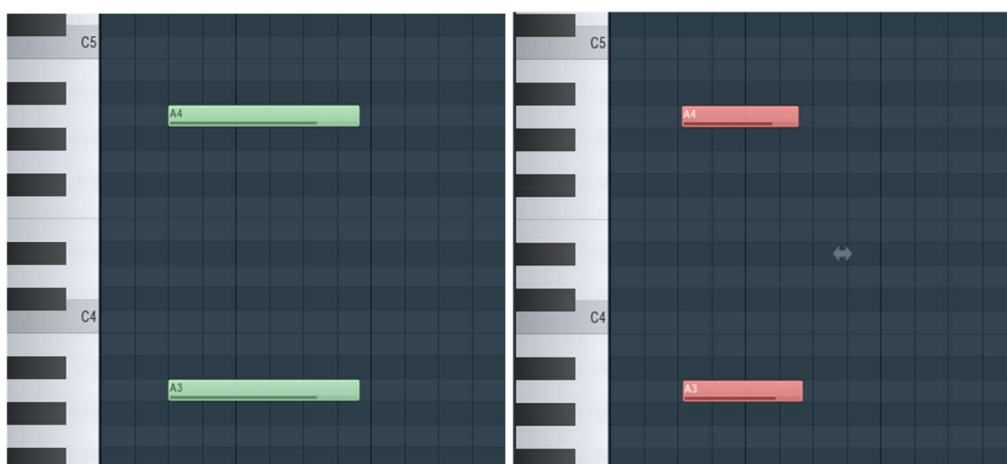


Рис. 5.1 – Исходный MIDI файл для эксперимента 1 и MIDI файл, полученный программой

Таблица 5.1

Характеристики оценки качества распознавания эксперимента 1

Характеристика	Значение
Относительное количество нот, совпавших в распознаваемом и оригинальном произведениях	100%

Продолжение табл. 5.1

Среднее относительное отклонение времени начала звучания каждой ноты	2,73%
Среднее относительное отклонение времени продолжительности звучания каждой ноты	38,18%
Относительное количество распознанных нот	100%
Относительное количество нераспознанных нот	0%

Алгоритм показал хорошие результаты в определении времени начала ноты, однако плохие в определении продолжительности, хотя ноты аккорда имеют примерно одинаковую длительность, что хорошо. Все распознанные ноты присутствуют в исходном сигнале, лишних нот (гармонических и субгармонических ошибок) нет, нераспознанных нот нет.

Эксперимент 2. Визуализация исходного и полученного MIDI-файлов представлена на рис. 5.2. Оценка качества распознавания представлена в табл. 5.2.

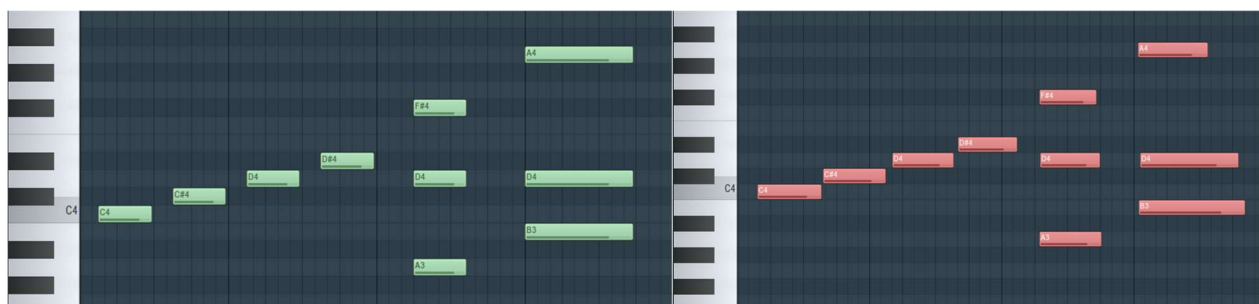


Рис. 5.2 – Исходный MIDI файл эксперимента 2 и MIDI файл, полученный программой

Таблица 5.2

Характеристики оценки качества распознавания эксперимента 2

Характеристика	Значение
Относительное количество нот, совпавших в распознаваемом и оригинальном произведениях	100%
Среднее относительное отклонение времени начала звучания каждой ноты	9,75%
Среднее относительное отклонение времени продолжительности звучания каждой ноты	23,67%
Относительное количество распознанных нот	100%
Относительное количество нераспознанных нот	0%

Алгоритм показал хорошие результаты в определении времени начала ноты и неплохие в определении продолжительности, хотя длительности нот аккорда не совпадают. Все распознанные ноты присутствуют в исходном сигнале, лишних нот (гармонических и субгармонических ошибок) нет, нераспознанных нот нет.

Эксперимент 3. Визуализация исходного и полученного MIDI-файлов представлена на рис. 5.3. Оценка качества распознавания представлена в табл. 5.3.



Рис. 5.3 – Исходный MIDI файл эксперимента 3 и MIDI файл, полученный программой

Таблица 5.3

Характеристики оценки качества распознавания эксперимента 3

Характеристика	Значение
Относительное количество нот, совпавших в распознаваемом и оригинальном произведениях	95%
Среднее относительное отклонение времени начала звучания каждой ноты	4,38%
Среднее относительное отклонение времени продолжительности звучания каждой ноты	31,59%
Относительное количество распознанных нот	95%
Относительное количество нераспознанных нот	5%

Алгоритм показал хорошие результаты в определении времени начала нот и плохие в определении продолжительности. Длительности нот некоторых аккордов не совпадают. Алгоритм плохо распознал пять одновременно играющих нот и не смог распознать одну из играющих нот – не смог распознать одну из них, а также в этом месте возникли трудности с определением длительности и начала нот. Все распознанные ноты присутствуют в исходном сигнале, лишних нот (гармонических и субгармонических ошибок) нет, одна нераспознанная нота.

Эксперимент 4. Визуализация исходного и полученного MIDI-файлов представлена на рис. 5.4. Оценка качества распознавания представлена в табл. 5.4.

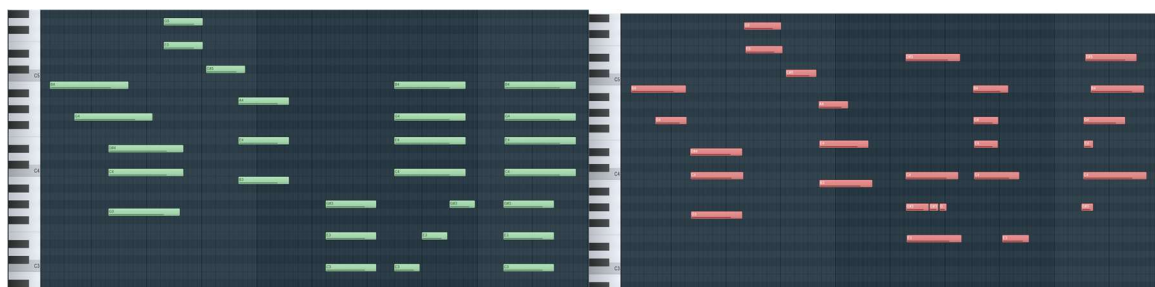


Рис. 5.4 – Исходный MIDI файл для эксперимента 3 и MIDI файл, полученный программой

Таблица 5.4

Характеристики оценки качества распознавания эксперимента 4

Характеристика	Значение
Относительное количество нот, совпавших в распознаваемом и оригинальном произведениях	82,14%
Среднее относительное отклонение времени начала звучания каждой ноты	3,32%
Среднее относительное отклонение времени продолжительности звучания каждой ноты	27,92%
Относительное количество распознанных нот	100%
Относительное количество нераспознанных нот	17,86%

Алгоритм показал хорошие результаты в определении времени начала нот и плохие в определении продолжительности. Длительности нот многих аккордов не совпадают. Алгоритм плохо распознал пять и шесть одновременно играющих нот – распознал на одну меньше, а также в этом месте возникли трудности с определением длительности и начала нот. Не все распознанные ноты присутствуют в исходном сигнале, лишние ноты (гармонические и субгармонические ошибки) есть, нераспознанные ноты есть.

Сумма относительного количества распознанных и нераспознанных нот превышает 100%, так как количество распознанных нот получилось равным количеству нот в исходном сигнале, но не все из них являются верными.

Проанализировав результаты работы программы, можно сказать, что алгоритм хорошо справляется с поиском начала звучания нот, но плохо справляется с определением их длительности. На определенную длительность сильно влияет начало других нот: если во время звучания ноты нажать еще и другую, алгоритм скорее всего оборвет предыдущую ноту здесь из-за сильной разнице в отклике НВП на всех масштабах в этот момент.

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены различные методы автоматической транскрипции музыкального произведения, изучен и применен метод непрерывного вейвлет-преобразования в задаче транскрипции музыкального сигнала.

В результате работы были решены следующие задачи:

- Создание материнского вейвлета и семейства вейвлетов по фрагменту сигнала;
- Применение алгоритма непрерывного вейвлет-преобразования;
- Разработка алгоритма транскрипции музыкального файла;
- Оценка качества работы алгоритма.

С помощью данного алгоритма можно распознавать отдельные ноты в мелодиях на пианино, определять время их начала и длительность. На данный момент алгоритм не может определять начальную громкость каждой ноты и присваивает всем нотам одинаковое значение.

Алгоритм показал хорошие результаты. Ноты несложных мелодий (до четырех нот, звучащих одновременно) хорошо распознаются: точно определяются основные частоты, не возникают гармонические и субгармонические ошибки. При большем количестве одновременно звучащих нот возникают трудности. Алгоритм хорошо справляется с определением начала звучания ноты, но имеет трудности с определением ее длительности – погрешность составляет минимум 20% и зависит от нажатия других нот во время ее звучания.

У данной работы есть перспективы развития:

1. Автоматизация. В ходе работы пришлось подбирать коэффициенты для фильтров вручную, для нового сигнала часто нужны были правки. Возможно автоматизировать этот процесс с помощью нейронной сети или других методов.

2. Расширить возможности распознавания. Добавить больше возможных музыкальных инструментов. Добавить возможность сначала записать с микрофона одну ноту, выбранную как базовую, чтобы затем записывать музыкальный сигнал с микрофона и распознавать игру этого инструмента.
3. Добавить другие формы нотации. Сейчас доступен только MIDI-файл, но возможно добавить и нотный стан.
4. Тестирование. Необходимо протестировать работу алгоритма на куда большем наборе данных, чтобы получить точную статистику и оценку. Сейчас это было делать нецелесообразно ввиду отсутствия автоматизации настройки коэффициентов фильтров. Так как их приходилось для каждого сигнала настраивать вручную для лучшего результата, тестирование на большем наборе данных представляется затруднительным, а статистика – необъективной.

Полученная программа представляет пользу ввиду необходимости автоматизации процесса ручной транскрипции, так как такой способ ненадежен и может привести к ошибке. Автоматическая транскрипция обеспечит быстрое, надежное и точное решение для транскрипции музыкальных инструментальных нот.

СПИСОК ЛИТЕРАТУРЫ

1. Reis G., Fonseca N., Ferndandez F. Genetic algorithm approach to polyphonic music transcription //2007 IEEE International Symposium on Intelligent Signal Processing. – IEEE, 2007. – С. 1-6.
2. Klapuri A., Virtanen T. Automatic music transcription //Handbook of Signal Processing in Acoustics. – Springer, New York, NY, 2008. – P. 277-303.
3. Benetos E. et al. Automatic music transcription: Breaking the glass ceiling. – 2012.
4. Фадеев, А. С. Идентификация музыкальных объектов на основе непрерывного вейвлет-преобразования: дис. кандидата технических наук. Томский политехнический университет, Томск, 2008.
5. Фадеев, А. С. Выбор вейвлет-функций для анализа музыкальной информации / А. С. Фадеев, Е. А. Кочегурова // Современные техника и технологии: Труды XII Междунар. научно-практ. конф. студентов и молодых ученых — г. Томск, 26—30 мар. 2007 г. — Томск: Изд-во ТПУ, 2006 — Т. 2. — С. 194—196
6. Фадеев, А. С. Формирование вейвлет-функций в задаче идентификации музыкальных сигналов // Известия Томского политехнического университета. — 2007. — Т. 311. — № 5. — С. 81—86.
7. Benson, D. Music: A Mathematical Offering / D. Benson. — Cambridge University Press, 2007. — 536 p
8. Polyphonic instrument recognition using spectral clustering / L. Martins, J. Burred, G. Tzanetakis, M. Lagrange // Proc. of the The International Conferences on Music Information Retrieval and Related Activities. — Vienna (Austria) 2007
9. Brown, Judith C. Musical fundamental frequency tracking using a pattern recognition method //The Journal of the Acoustical Society of America. 1992. Vol. 92(3). P. 1394-1402

ПРИЛОЖЕНИЕ 1

Код программы для непрерывного вейвлет-преобразования

```

from scipy.signal import correlate
from scipy.ndimage import median_filter
from scipy.ndimage import maximum_filter

def my_cwt(data, scales):
    # scales:
    # Получаем на вход исходный сигнал и список масштабов, или нот,
    # существование которых нужно проверить в сигнале
    if type(scales) != list and type(scales) != np.ndarray:
        #если вдруг в scales решили послать не список
        scales = [scales]
    cwt_result = np.zeros((len(scales), len(data)))
    # каждая строка - результат взаимнокорреляционной функции
    # для вейвлета с этим масштабом
    for i in range(len(scales)):
        wave = conv_piano_wavelet(scales[i])
        if len(wave)>len(cwt_result[i]):
            #если длина ноты (пусть и 16 периодов) больше длины сигнала, то ее т
            ам точно нет
            cwt_result[i] = [0]*len(cwt_result[i])
        else:
            #Рассчитываем Wf_s - результат взаимной корреляции между исходным си
            гналом и w_s
            cwt_result[i] = correlate(data, wave, mode='same', method='fft')
            modifier = np.power(abs(np.power(2, (-
            scales[i]/12), dtype=np.float64))), -0.5)
            #Умножаем Wf_s на |modifier|^(-1/2) согласно формуле (3.9)
            cwt_result[i] *= modifier

    #фильтры
    filtered_cwt = np.zeros(cwt_result.shape)
    for line_num in range(len(cwt_result)):
        # 1.Возьмем абсолютные значения
        # 2.Фильтр максимума по строке;
        # 3.Медианный фильтр по строке;
        filtered_cwt[line_num] = median_filter(maximum_filter(abs(cwt_result[line_num]), 401), 401)
    return filtered_cwt

```

ПРИЛОЖЕНИЕ 2

Код программы для эксперимента с синусоидальным сигналом

```
import numpy as np
import matplotlib.pyplot as plt

#задаем частоту дискретизации
SAMPLING_FREQUENCY = 44100

#функция для синтеза синусоидального сигнала
def synthesize_signal(stop_time, *frequencies):
    '''stop_time - длительность, сек\n
    sampling_frequency - частота дискретизации\n
    frequencies - частоты, которые будем складывать'''

    if not frequencies: raise AttributeError
    array_length = int(SAMPLING_FREQUENCY * stop_time) + 1
    test_signal = np.zeros(array_length)
    time_array = np.linspace(0, stop_time, array_length)
    for frequency in frequencies:
        test_signal += np.sin(2*np.pi*frequency*time_array)
    return test_signal

#используемые основные частоты.
Fc1 = 261.63 #нота C4
Fc2 = 329.63 #нота E4
Fc3 = 392 #нота G4
#создаем тестовый сигнал
test_signal = synthesize_signal(0.2, Fc1, Fc2, Fc3)
for note_freq in notes_original.keys():
    print(f'{note_freq} Hz: {notes_original[note_freq]}')
stop_time = 0.2
time_array = np.linspace(0, stop_time, len(test_signal))

def conv_piano_wavelet(scale, periods):
    #center frequency базовой ноты. 440 - нота ля,
    # по ней обычно настраивают инструменты
    n0_freq = 440
    #коэффициент масштабирования
    #  $2^{(-i/12)}$ , где  $i$  - положение ноты  $N_i$  по высоте относительно ноты  $N_0$ 
    #  $i$  целое
    modifier = np.power(2, (-scale/12), dtype=np.float64)
    #если построить вейвлет на  $[0, \text{length}]$ , в нем будет {periods} периодов
    length = periods*modifier/n0_freq #seconds
    mother_wavelet = lambda t: np.sin(2*np.pi*n0_freq*(t/modifier))
    x = np.linspace(-length/2, length/2, int(SAMPLING_FREQUENCY*length))
    res = mother_wavelet(x)
    return res
```

```

widths = list(np.arange(-36,36.01))
cwt_result = my_cwt(test_signal,widths)

def vertical_filter(data):
    new_matrix = np.zeros(data.shape)
    for column_num in range(len(test_signal)):
        local_threshold = max(data[:,column_num]) * 0.4
        new_matrix[:,column_num] = [item if item>local_threshold else 0 for item
in data[:,column_num]]
    return new_matrix

# фильтр по порогу
trimmed_cwt = vertical_filter(cwt_result)
# локальный вертикальный фильтр
bin_cwt = local_vertical_maximum_filter(trimmed_cwt)

def count_frequency(scale):
    n0 = 440
    modifier = np.power(2, (-scale/12), dtype=np.float64)
    return round(n0/modifier,2)

```

ПРИЛОЖЕНИЕ 3

Код собственных фильтров, использованных в программе

Фильтр порога по максимуму

```
def threshold_max_filter(dataId, percentage) -> list:
    assert percentage<1 and percentage>0
    threshold = max(dataId) * percentage
    return [item if item > threshold else 0 for item in dataId]
```

Фильтр локального максимума

```
def local_vertical_maximum_filter(data):
    new_matrix = np.zeros(data.shape, dtype=int)
    for column_num in range(len(test_signal)):
        column = data[:,column_num]
        for cell_num in range(len(column)):
            if cell_num>1 and cell_num<(len(column)-3):
                new_matrix[cell_num,column_num] = 1 if (column[cell_num]>column[cell_num-1] and column[cell_num]>column[cell_num-2] and column[cell_num]>column[cell_num+1] and column[cell_num]>column[cell_num+2]) else 0
            elif cell_num<=1:
                new_matrix[cell_num,column_num] = 1 if (column[cell_num]>column[cell_num+1] and column[cell_num]>column[cell_num+2]) else 0
            elif cell_num>=(len(column)-3):
                new_matrix[cell_num,column_num] = 1 if (column[cell_num]>column[cell_num-1] and column[cell_num]>column[cell_num-2]) else 0
    return new_matrix
```

Вертикально-октавный фильтр

```
def octave_filter(cwt_data):
    #фильтруем вертикально-октавно
    if len(widths)>13: #чтобы там вообще были октавы
        rows = list(range(len(widths))) #0->len(widths)
        cwt_vertical_octave_filtered = np.zeros_like(cwt_data)
        for column_num in range(len(cwt_data[0])):
            column = list(cwt_data[:,column_num])
            for row_num in range(0,12):
                #возьмем все значения через октаву
                octave_row_nums = rows[row_num::12]
                if len(octave_row_nums)>1:
                    octave_values = np.take(cwt_data[:,column_num], octave_row_nums)
                    #останутся только самые сильные значения
                    octave_threshold = max(octave_values) * 0.95
```

```

        octave_values_filtered = [item if item > octave_threshold el
se 0 for item in octave_values]
        for row_num, value in zip(octave_row_nums, octave_values_fil
tered):
            column[row_num] = value
            cwt_vertical_octave_filtered[:,column_num] = column
        return cwt_vertical_octave_filtered
    else:
        return cwt_data

```

ПРИЛОЖЕНИЕ 4

Код программы для поиска нотных объектов в результатах НВП синусоидального сигнала

```
def note_noter(data):
    notes = {}
    signal_length = len(data[0])
    min_length = 700
    for line_num in range(len(data)):
        #На строке найдем индексы начала цепочек
        # «единиц» (нота присутствует) и «нулей» (нота отсутствует).
        indexes = list(np.nonzero(np.r_[1, np.diff(data[line_num])[:-1]])[0])
        indexes.append(signal_length-1)
        #Получим длины цепочек повторений
        slices_lengths = list(abs(np.subtract(indexes[:-1], indexes[1:])))
        slices_signed = (([0, 1] if data[line_num][0]==0 else [1,0]) * int(len(indexes)/2))[:len(indexes)]
        idx = 0
        while idx<len(slices_signed):
            if slices_signed[idx] == 0 and slices_lengths[idx]<min_length:
                del(slices_signed[idx])
                del(slices_lengths[idx])
                del(indexes[idx])
                continue
            idx +=1
        idx = 1
        while idx<len(slices_signed):
            if slices_signed[idx] == 1 and slices_signed[idx-1]==1:
                slices_lengths[idx-1] += slices_lengths[idx]
                del(slices_signed[idx])
                del(slices_lengths[idx])
                del(indexes[idx])
                continue
            idx+=1
        for (sign, idx, length) in zip(slices_signed,indexes,slices_lengths):
            if sign==1 and length >= min_length:
                if count_frequency(widths[line_num]) not in notes:
                    notes[count_frequency(widths[line_num])] = []
                notes[count_frequency(widths[line_num])].append([idx, length])
    return notes

notes = note_noter(bin_cwt)
```

ПРИЛОЖЕНИЕ 5

Код программы для эксперимента с сигналом, приближенным к реальному

```
#получаем материнские вейвлеты
#материнский вейвлет OnlinePiano
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import read
real_note = read('audio/onlinepiano/A4_trim.wav')
SAMPLING_FREQUENCY = real_note[0]
start = 52000
ending = start + 1751
real_note = real_note[1]
real_wave = real_note[start:ending]
print(f'Интеграл: {np.sum(real_wave)}')
#обрабатываем, чтобы интеграл стал 0
from scipy.fft import fft, ifft
fft_wave = fft(real_wave)
fft_wave[0] = 0 #первое число это интеграл!
ifft_wave = ifft(fft_wave)
print(f'Интеграл: {np.sum(np.real(ifft_wave))}')
onlinepiano_wavelet = np.real(ifft_wave)
#материнский вейвлет OnlinePiano (FL Studio)
real_note = read('audio/flstudio/A4.wav')
SAMPLING_FREQUENCY = real_note[0]
real_note = real_note[1]
start = 87869
ending = start + 3161
real_wave = real_note[start:ending]
print(np.sum(real_wave))
#обрабатываем, чтобы интеграл стал 0
fft_wave = fft(real_wave)
fft_wave[0] = 0 #первое число это интеграл!
ifft_wave = ifft(fft_wave)
print(f'Интеграл: {np.sum(np.real(ifft_wave))}')
flstudio_wavelet = np.real(ifft_wave)
#кусочно-линейная аппроксимация
def piecewise_linear_wavelet(mother_wavelet):
    def res(t):
        if t < -1 or t >= len(mother_wavelet)-1:
            return 0
        elif t < 0:
            return mother_wavelet[0]*abs(1-t)
        else:
            i = int(np.trunc(t))
            #аппроксимация
            return mother_wavelet[i] + (mother_wavelet[i+1] - mother_wavelet[i])
    * (t-i)
    return res
```

```

def mother_wavelet_approximation(x):
    return piecewise_linear_wavelet(mother_wavelet)(x)
#нулевые, чтобы потом выбрать материнский вейвлет: onlinepiano или flstudio
mother_wavelet = None
mother_wavelet_approximation_length = 0
def conv_piano_wavelet(scale):
    '''возвращает дочерний вейвлет с заданным scale - масштабом'''
    modifier = np.power(2, (-scale/12), dtype=np.float64)
    #масштабируем материнский вейвлет
    # с помощью кусочно-линейной аппроксимации
    # точка t дочки соответствует точке t/modifier матери
    daughter_wavelet = lambda t: mother_wavelet_approximation(t/modifier)
    #длина матери * модификатор = длина ребенка. массив иксов длины ребенка.
    x = list(range(int(mother_wavelet_approximation_length*modifier)))
    #применяем функцию daughter_wavelet на иксы.
    return np.array(list(map(daughter_wavelet,x)))
#возможность выбрать материнский вейвлет
piano_wavelets = ['onlinepiano', 'flstudio']
def choose_wavelet_origin(origin:str):
    assert origin in piano_wavelets, f"No wavelet with such origin! Available wa
velets: {piano_wavelets}"
    global mother_wavelet, mother_wavelet_approximation_length
    if origin=='onlinepiano':
        mother_wavelet = onlinepiano_wavelet
        mother_wavelet_approximation_length = len(mother_wavelet)
    elif origin=='flstudio':
        mother_wavelet = flstudio_wavelet
        mother_wavelet_approximation_length = len(mother_wavelet)

#получаем исходный сигнал
from scipy.io.wavfile import read
folder = 'audio/flstudio/'
file_name = 'test05'
file_format = '.wav'
test_signal = read(folder+file_name+file_format)
SAMPLING_FREQUENCY = test_signal[0]
test_signal = test_signal[1]
choose_wavelet_origin('flstudio')
widths = list(np.arange(-24,12.01))
cwt_result = my_cwt(test_signal,widths)
cwt_amplified = cwt_result
#вертикально-октавный фильтр
cwt_vertical_octave_filtered = octave_filter(cwt_amplified)
#фильтруем по порогу столбцы
cwt_vertical_filtered = np.zeros_like(cwt_vertical_octave_filtered)
for column_num in range(len(cwt_vertical_filtered[0])):
    column_max_threshold = max(cwt_vertical_octave_filtered[:,column_num]) * 0.3
    cwt_vertical_filtered[:,column_num] = [item if item > column_max_threshold e
lse 0 for item in cwt_vertical_octave_filtered[:,column_num]]

```


ПРИЛОЖЕНИЕ 6

Код программы для поиска нотных объектов в результатах НВП сигнала,
приближенного к реальному

```
#возьмем производную, чтобы найти начала нот
def note_beginnings(cwt_row) -> np.ndarray:
    #получаем производную и фильтруем пики
    cwt_row_diff = abs(np.diff(cwt_row))
    cwt_row_diff_threshold = max(cwt_row_diff) * 0.25
    cwt_row_diff_bin = [1 if item > cwt_row_diff_threshold else 0 for item in cwt_row_diff]
    #теперь ищем начала заметных единиц
    cwt_row_diff_bin_indices = list(np.nonzero(np.r_[1, np.diff(cwt_row_diff_bin)[:-1]])[0])
    lengths = cwt_row_diff_bin_indices.copy()
    min_interval = 5000 #минимальный интервал между началами нот
    if cwt_row_diff_bin[0] == 0:
        lengths[::2] = [0]*len(cwt_row_diff_bin_indices[::2])
    else:
        lengths[1::2] = [0]*len(cwt_row_diff_bin_indices[1::2])
    #получаем длины между единицами
    lengths_between_ones = np.append(lengths[2:], np.subtract(lengths[2:], lengths[:-2]))
    lengths_good_indices = np.where(lengths_between_ones > min_interval)[0]
    #возвращаем начала пиков
    return np.take(cwt_row_diff_bin_indices, lengths_good_indices)

def count_frequency(scale):
    n0 = 440
    modifier = np.power(2, (-scale/12), dtype=np.float64)
    return round(n0/modifier,2)

def note_noter(cwt_data) -> dict:
    MIN_LENGTH = 8000 #минимальная длина ноты
    rows_maxes = [max(row) for row in cwt_data]
    NOTE_EXISTANCE_THRESHOLD = max(rows_maxes) * 0.44
    noted_rows = np.array([1 if row_max>=NOTE_EXISTANCE_THRESHOLD else 0 for row_max in rows_maxes])
    noted_rows_indices = np.where(noted_rows==1)[0]
    print(noted_rows_indices)

    cwt_data_filtered = np.zeros_like(cwt_data)
    for row in noted_rows_indices:
        cwt_data_filtered[row] = median_filter(maximum_filter(cwt_data[row],401),401)
    plt.imshow(cwt_data_filtered, aspect='auto', interpolation='none',
               extent=[0, cwt_data_filtered.shape[1]-1, widths[-1], widths[0]])
```

```

plt.show()

notes = {}
for row in noted_rows_indices:
    freq = count_frequency(widths[row])
    beginnings = note_beginnings(cwt_data_filtered[row])

    for note_beginning_num in range(len(beginnings)):
        if note_beginning_num+1 >= len(beginnings):
            interval_end = len(cwt_data[0])-1
        else:
            # если следующего начала ноты нет
            # концом интервала будет конец сигнала
            interval_end = beginnings[note_beginning_num+1]
        interval = cwt_data_filtered[row][beginnings[note_beginning_num]:interval_end]

        #порог для остановки ноты
        interval_max = max(interval)
        interval_threshold = interval_max * 0.06

        #ищем место максимума в интервале - старт ноты
        for interval_idx in range(len(interval)):
            if interval[interval_idx] == interval_max:
                break
        note_start = interval_idx

        #ищем ноль (место меньше interval_threshold) - конец ноты
        for interval_idx in range(note_start, len(interval)):
            if interval[interval_idx] < interval_threshold:
                break
        note_end = interval_idx

        #длина ноты
        note_length = note_end - note_start
        if note_length < MIN_LENGTH: continue

        #переведем начало и конец ноты в систему отсчета всего сигнала
        note_start += beginnings[note_beginning_num]
        note_end += beginnings[note_beginning_num]
        note_info = {
            'start': note_start,
            'end': note_end,
            'length': note_length
        }
        if freq not in notes: notes[freq] = []
        notes[freq].append(note_info)
    return notes

notes = note_noter(cwt_vertical_filtered)

```

ПРИЛОЖЕНИЕ 7

Код программы для записи распознанных нот в MIDI файл и сравнения с исходным MIDI файлом

```
import pretty_midi
from copy import deepcopy

def frequency_to_midi_number(frequency):
    #A4 - это №69
    return round(12*(np.log2(frequency) - np.log2(440.0)) + 69)

tempo = 130
MyMIDI = pretty_midi.PrettyMIDI(initial_tempo=tempo)
piano = pretty_midi.Instrument(program=0, is_drum=False, name='Acoustic Grand Piano')
MyMIDI.instruments.append(piano)

notes_s = deepcopy(notes) #ноты в секундах
for note_freq in notes_s.keys():
    for note_data in notes_s[note_freq]:
        note_data["start"] = round(note_data["start"] / SAMPLING_FREQUENCY, 6)
        note_data["end"] = round(note_data["end"] / SAMPLING_FREQUENCY, 6)
        note_data["length"] = round(note_data["length"] / SAMPLING_FREQUENCY, 6)

for note_freq in notes_s.keys():
    # Retrieve the MIDI note number for this note name
    note_number = frequency_to_midi_number(note_freq) #баг библиотеки
    for note_data in notes_s[note_freq]:
        # Create a Note instance for this note, starting at START seconds and ending at END seconds
        note = pretty_midi.Note(velocity=100, pitch=note_number-12, start=note_data["start"], end=note_data["end"])
        # Add it to our piano instrument
        piano.notes.append(note)

MyMIDI.write('result_midi.mid')
MyMIDI.write(folder+file_name+'_result.mid')

# прочитаем исходный миди
original_midi = pretty_midi.PrettyMIDI(folder+file_name+'.mid')
origin_notes = deepcopy(original_midi.instruments[0].notes)
my_notes = deepcopy(MyMIDI.instruments[0].notes)

def is_identical(note1, note2):
    return (note1.start == note2.start and
            note1.end == note2.end and
            note1.pitch == note2.pitch and
            note1.velocity == note2.velocity)
```

```

def remove_duplicates(sorted_notes:list):
    note_idx = 0
    #проверяем на идентичность только соседние ноты, тк все отсортировано по ста
    рту
    while note_idx < len(sorted_notes)-1:
        if is_identical(sorted_notes[note_idx], sorted_notes[note_idx+1]):
            del(sorted_notes[note_idx])
        else:
            note_idx += 1

remove_duplicates(origin_notes)
remove_duplicates(my_notes)

origin_notes_amount = len(origin_notes)
my_notes_amount = len(my_notes)

#для удобства рассортируем ноты оригинала и мои ноты
# в словарь по используемой ноте
origin_notes_pitched = {}
for note_data in origin_notes:
    if note_data.pitch not in origin_notes_pitched:
        origin_notes_pitched[note_data.pitch] = []
    origin_notes_pitched[note_data.pitch].append(note_data)
my_notes_pitched = {}
for note_data in my_notes:
    if note_data.pitch not in my_notes_pitched:
        my_notes_pitched[note_data.pitch] = []
    my_notes_pitched[note_data.pitch].append(note_data)

# ключ: оригинальная нота.
# значение: нота из результата, которая с ней пересекается,
# если такая есть, иначе None.
paired_notes = {}
for note_data in origin_notes:
    #массив на всякий случай. вдруг моя прога разобьет ноту на две
    paired_notes[note_data] = []

my_leftovers = [] #мои ноты, которым не нашлось места

def overlap(note1, note2) -> bool:
    #ноты пересекаются?
    if note2.start > note1.end or note1.start > note2.end:
        return False
    return True

for pitch in my_notes_pitched.keys():
    my_notes_of_one_pitch = my_notes_pitched[pitch]
    origin_notes_of_one_pitch = origin_notes_pitched[pitch]
    for my_note in my_notes_of_one_pitch:

```

```

    for origin_note in origin_notes_pitched[pitch]:
        if overlap(my_note, origin_note):
            paired_notes[origin_note].append(my_note)
            break
    else:
        #не нашли оригинальную пару для моей ноты
        my_leftovers += my_note

#считаем метрики
amount_of_paired_notes = sum([1 if len(obj)>0 else 0 for obj in paired_notes.values()])
relative_amount_of_paired_notes = round(amount_of_paired_notes / origin_notes_amount * 100, 2)
print(f'Относительное количество нот, совпавших в распознаваемом и оригинальном произведениях: {relative_amount_of_paired_notes} %')

def calculate_relative_diff_start():
    res = 0
    for origin_note in paired_notes.keys():
        if len(paired_notes[origin_note])==0:
            continue
        origin_note_length = origin_note.end - origin_note.start
        diff_start = abs(origin_note.start - paired_notes[origin_note][0].start)
        res += diff_start/origin_note_length
    return res

relative_diff_start = round(calculate_relative_diff_start()/origin_notes_amount*100,2)
print(f'Среднее относительное отклонение времени начала звучания каждой ноты: {relative_diff_start} %')

def calculate_relative_diff_length():
    res = 0
    for origin_note in paired_notes.keys():
        if len(paired_notes[origin_note])==0:
            continue
        origin_note_length = origin_note.end - origin_note.start
        my_note_length = paired_notes[origin_note][0].end - paired_notes[origin_note][0].start
        diff_length = abs(origin_note_length - my_note_length)
        res += diff_length/origin_note_length
    return res

relative_diff_length = round(calculate_relative_diff_length()/origin_notes_amount*100,2)
print(f'Среднее относительное отклонение времени продолжительности звучания каждой ноты: {relative_diff_length} %')

relative_amount_of_recognized_notes = round(my_notes_amount/origin_notes_amount*100, 2)

```

```
print(f'Относительное количество распознанных нот: {relative_amount_of_recognize  
d_notes} %')  
  
amount_of_unrecognized_notes = sum([1 if len(obj)==0 else 0 for obj in paired_no  
tes.values()])  
relative_amount_of_unrecognized_notes = round(amount_of_unrecognized_notes / ori  
gin_notes_amount * 100, 2)  
print(f'Относительное количество нераспознанных нот: {relative_amount_of_unrecog  
nized_notes} %')
```