«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	09.03.01 – Информатика и вычислительная техника
Профиль	Вычислительные машины, комплексы, системы и сети
Факультет	Компьютерных технологий и информатики
Кафедра	Вычислительной техники

К защите допустить

Зав. кафедрой

д. т. н, профессор М. С. Куприянов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: ВЕРИФИКАЦИЯ МУЗЫКАЛЬНОГО ПРОИЗВЕДЕНИЯ

Студент	_____	М. А. Чайкин
Руководитель к. ф.-м. н., доцент	_____	С. В. Рыбин
Консультант к. т. н., доцент	_____	И. С. Зуев
Консультант к. э. н., доцент	_____	М. А. Косухина

Санкт-Петербург
2021

Санкт-Петербургский государственный электротехнический университет "ЛЭТИ" им. В. И. Ульянова (Ленина)
(СПбГЭТУ "ЛЭТИ")

09.03.01 – Информатика и вычислительная техника
Профиль Вычислительные машины, комплексы, системы и сети
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
М. С. Куприянов
"___" _____ 20__г.

ЗАДАНИЕ
на выпускную квалификационную работу

Студент М. А. Чайкин Группа № **7305**

1. Тема

Верификация музыкального произведения.

Место выполнения ВКР: Кафедра алгоритмической математики.

2. Объект и предмет исследования

Объект исследования – схожесть музыкальных произведений. Предмет исследования – программная реализация вычисления метрики схожести музыкальных произведений.

3. Цель

Построение метрики для определения схожести музыкальных произведений.

4. Исходные данные

Исходными данными являются англоязычные и русскоязычные ресурсы в сети интернет, научные статьи, а также документации к средствам разработки.

5. Содержание

Разработка алгоритма вычисления метрики схожести музыкальных произведений, сравнительный анализ его модификаций и выбор наиболее корректно работающего варианта, а также программная реализация вычисления метрики схожести.

6. Технические требования

Метрика должна отражать схожесть звучания музыкальных произведений и быть однозначной для любой пары композиций. Метрика должна быть тем меньше, чем более схожи музыкальные произведения, и, соответственно, тем больше, чем менее схожи музыкальные произведения.

7. Дополнительные разделы

Разработка и стандартизация программных средств.

8. Результаты

Построенная метрика определения схожести музыкальных произведений и программная реализация ее вычисления.

Дата выдачи задания
«26» февраля 2021 г.

Дата представления ВКР к защите
«17» июня 2021 г.

Студент _____ М. А. Чайкин

Руководитель
к. ф.-м. н., доцент _____ С. В. Рыбин

Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление (специальность)
Профиль (программа, специализация)
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
“___” _____ 202__г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

Тема **Верификация музыкального произведения**

Студент М. А. Чайкин Группа № **7305**

№ этапа	Наименование работ	Срок выполнения
1	Изучение теоретической части по работе с музыкальными произведениями	15.03 – 04.04
2	Ознакомление с библиотеками и инструментами выбранного языка программирования	05.04 – 10.04
3	Разработка алгоритма определения схожести музыкальных произведений	11.04 – 30.04
4	Разработка программного кода	04.05 – 10.05
5	Тестирование разработанной программы	11.05 – 14.05
6	Оформление пояснительной записки	15.05 – 06.06
7	Предварительное рассмотрение работы	07.06 – 08.06
8	Представление работы к защите	17.06

Студент _____ М. А. Чайкин

Руководитель к. ф.-м. н., доцент _____ С. В. Рыбин

РЕФЕРАТ

Пояснительная записка к выпускной квалификационной работе изложена на 76 стр., состоит из 5 разделов и включает 35 рисунков, 5 таблиц, 12 ссылок на литературные источники и 1 приложение.

Целью работы является построение метрики для определения схожести музыкальных произведений.

В выпускной квалификационной работе описаны существующие приложения, которые для решения поставленных перед ними задач используют методы определения схожести музыкальных композиций. Также в работе представлен алгоритм вычисления значения метрики схожести музыкальных произведений, сравнительный анализ его модификаций и их тестирование.

Результатом работы является разработанный алгоритм и его программная реализация, на вход которой подаются два музыкальных произведения, в качестве результата работы программа отображает значение метрики схожести для поданных на вход музыкальных композиций.

ABSTRACT

The aim of the work is to build a metric to determine the similarity of musical works.

In the final qualifying work, existing applications are described, which, to solve the tasks assigned to them, use methods for determining the similarity of musical works. In addition, the paper presents algorithm for calculating the value of the similarity metric of musical works, comparative analysis of its modifications and their testing.

The result of the work is a developed algorithm and application, the input of which is two pieces of music; as a result of the work, the program displays the value of the similarity metric for the input of musical compositions.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяют следующие термины с соответствующими определениями:

ДПФ – дискретное преобразование Фурье.

ПС – программное средство.

ОНК – отфильтрованный набор кадров.

МНС – матрица нотного спектра.

ПРВ – плотность распределения вероятностей.

МПРВ – матрица плотности распределения вероятностей.

МКЛ – метрика Кульбака-Лейблера.

DKL(P, Q) – расстояние Кульбака-Лейблера между распределениями вероятностей P и Q, где Q является приближением P.

ВВЕДЕНИЕ

Актуальность данной работы обусловлена широкой практикой применения определения схожести звуков в разных сферах жизни. Например, биологи, занимаясь исследованием звуков, издаваемых животными, используют их акустические отпечатки для последующего анализа. Современные широко применяемые мобильные приложения, которые определяют название музыкальной композиции в реальном времени по ее звучанию, используют алгоритмы с анализом схожести музыкальных треков. Социальные сети классифицируют музыку по ее схожести для оптимизации хранимых данных. Приложения для прослушивания музыки формируют рекомендации пользователям на основе схожести с музыкальными композициями, прослушиваемыми данными пользователями.

Целью работы является построение метрики для определения схожести музыкальных произведений.

Объектом исследования работы является схожесть музыкальных произведений. Предметом исследования работы является программная реализация вычисления метрики схожести музыкальных композиций.

В первом разделе представлены теоретические основы работы, необходимые для описания разработанного алгоритма. Во втором разделе происходит обзор существующих решений по данной и схожей тематике. Третий раздел содержит полное изложение всех рассмотренных модификаций алгоритма определения схожести музыкальных композиций. В четвертом разделе происходит тестирование вариантов алгоритма, и делаются выводы по полученным экспериментально результатам. Пятый раздел содержит описание разработки и стандартизации ПС.

1 Теоретические основы определения схожести музыкальных произведений

В данном разделе представлены теоретические основы работы, необходимые для дальнейшего описания разработанного алгоритма определения метрики схожести между музыкальными произведениями.

1.1 Представление звукового сигнала

Сигнал является величиной, которая изменяется во времени. Звук – это вариация в давлении воздуха, соответственно, звуковой сигнал представляет собой изменение давления воздуха во времени [6]. В силу того, что звук является волной, его можно охарактеризовать спектром частот и соответствующими им значениями амплитуд.

Звуковые колебания, находящиеся в диапазоне частот от 20 Гц до 20 кГц, являются слышимыми для человека. Эмпирические данные свидетельствуют о том, что основная информация о звуковом сигнале (н-р, музыкальной композиции) располагается в диапазоне частот от 220 Гц до 4 кГц. Соответственно, для анализа музыкальных произведений необходимо в первую очередь проводить работу именно с данным частотным диапазоном.

Целью данной работы является построение метрики схожести музыкальных произведений. Стоит заметить, что понятие схожести весьма субъективно, и для разных людей величина схожесть одних и тех же музыкальных произведений может быть различной. Но несмотря на это, все же есть закономерности, которые описывают единую форму восприятия всеми людьми тех или иных раздражителей. Таким является эмпирический закон Вебера-Фехнера, который гласит о том, что интенсивность ощущения прямо пропорциональна логарифму интенсивности раздражителя. Именно от установки данного закона будет происходить анализ схожести музыкальных произведений и вычисление соответствующей метрики.

Зафиксируем натуральное число N. Будем называть сигналом N периодическую комплекснозначную функцию целочисленного аргумента $x = x(j) = x_j$, j ϵ Z. Можно считать, что задана последовательность комплексных чисел $x = \{x(j)\}_{j=0}^{N-1}$ продолженная периодически на все целые индексы, то есть $x_{j+sN} = x_j$, для любого s ∈ Z. Множество сигналов будем обозначать C_N [1].

Единичным N-периодическим импульсом называется сигнал δ_N, у которого δ_N(j) = 1, когда j делится на N, и δ_N(j) = 0 в остальных случаях.

Для любого сигнала $x \in C_N$ справедливо равенство:

$$x(j) = \sum_{k=0}^{N-1} x(k) * \delta_N(j-k), \qquad j \in Z$$

1.2 Преобразование Фурье

Преобразование Фурье [5] – это преобразование, которое ставит в соответствие каждой функции вещественной переменной $f(x)$ ее фурье-образ $F(f) = \hat{f}(\omega)$ согласно следующей формуле:

$$\hat{f}(w) = \int_{-\infty}^{+\infty} f(x) * e^{-2\pi i x \omega}\, dx$$

Преобразование по данной формуле называют непрерывным преобразованием Фурье, поскольку функция f(x) является непрерывной. Полученный фурье-образ также будет являться непрерывной функцией. Данное преобразование является обратимым, поэтому исходную функцию f(x) можно восстановить по ее фурье-образу $\hat{f}(w)$ согласно формуле:

$$f(x) = \int_{-\infty}^{+\infty} \hat{f}(\omega) * e^{2\pi i \omega x}\, d\omega$$

Такое преобразование называется обратным преобразованием Фурье.

Основные свойства преобразования Фурье:

1. Линейность – преобразование Фурье линейной комбинации функций равно линейной комбинации преобразований Фурье от каждой функции в отдельности:

$$F\big(\alpha * f(x) + \beta * g(x)\big) = \alpha * F\big(f(x)\big) + \beta * F\big(g(x)\big).$$

2. Зависимость преобразования Фурье от сдвига аргумента функции выражается следующим равенством:

$$F(f(x-a)) = e^{-2\pi i a\omega} * F(f(x)).$$

3. Сжатие (растягивание) исходной функции пропорционально сжимает (растягивает) её фурье-образ:

$$F\big(f(a*x)\big) = \frac{1}{|a|} * \hat{f}\left(\frac{\omega}{a}\right).$$

4. Преобразование Фурье свертки функций равно поточечному перемножению их фурье-образов:

$$F(f * g) = \hat{f}(\omega) * \hat{g}(\omega).$$

5. Преобразование Фурье поточечного преобразования функций равно свертке их фурье-образов:

$$F\big(f(x) * g(x)\big) = \big(\hat{f} * \hat{g}\big)(x).$$

6. Преобразование Фурье сохраняет “энергию” сигнала:

$$\int_{-\infty}^{+\infty} \left|\hat{f}(\omega)\right|^2 d\omega = \int_{-\infty}^{+\infty} |f(x)|^2 \, dx.$$

Преобразование Фурье часто применяется к функциям времени, которые называются сигналами, для определения их спектра, частотного состава. Полученный фурье-образ – функция частоты, каждое значение которой будет являться, в общем случае, комплексным числом, модуль которого является амплитудным значением для данной частоты, а аргумент – фазовым значением для данной частоты [5]. При таком подходе преобразование Фурье рассматривается как обратимое преобразование функции из временного пространства в частотное пространство. Полученный фурье-образ является ап-

проксимацией исходного сигнала суммой бесконечного числа гармоник разных частот и амплитудных значений.

Дискретное преобразование Фурье (ДПФ) – это отображение F_N: $C_N \rightarrow C_N$, сопоставляющее сигналу x сигнал $X = F_N(x)$ со значениями

$$X(k) = \sum_{j=0}^{N-1} x(j) * \omega_N^{-kj}, \qquad \text{где } k \in Z, \tag{1.1}$$

$$\omega_N = \cos\frac{2\pi}{N} + i * \sin\frac{2\pi}{N} = e^{\frac{2\pi i}{N}} \tag{1.2}$$

Сигнал X называется спектром Фурье сигнала x или просто спектром, а величины X(k) – компонентами спектра или спектральными составляющими.

Обратное ДПФ позволяет восстановить по X_k исходную функцию $x(j)$:

$$x(j) = \frac{1}{N} * \sum_{k=0}^{N-1} X(k) * \omega_N^{kj}, \qquad k \in Z.$$

Вычисление спектра X можно интерпретировать как вычисление значений полинома

$$P(z) = \sum_{j=0}^{N-1} x_j * z^j$$

в точках единичной окружности $z_k = \omega_N^k, k \in 0: N-1$, и как вычисление произведения матрицы

$$F_N = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_N & & \omega_N^{N-1} \\ & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \cdots & \omega_N^{(N-1)(N-1)} \end{pmatrix}$$

на вектор x.

Сигнал x называется четным если $x(-j) = \bar{x}(j)$, и нечетным, если $x(-j) = -\bar{x}(j)$ при всех $j \in Z$. Сигнал x называется вещественным, если $Im(x) = 0$, и чисто мнимым, если $Re(x) = 0$.

Основные свойства ДПФ:

1. Сигнал $x \in C_N$ является вещественным, тогда и только тогда, когда его спектр Фурье $X = F_N(x)$ четный.
2. Сигнал $x \in C_N$ является четным, тогда и только тогда, когда его спектр Фурье $X = F_N(x)$ – вещественный.
3. Пусть a и b – два вещественных сигнала из C_N. Образуем комплексный сигнал $x = a + ib$. Тогда для спектров A,B,X этих сигналов справедливы соотношения

$$A(k) = [X(k) + \bar{X}(N-k)],\ B(k) = -\frac{1}{2}i[X(k) - \bar{X}(N-k)]$$

при всех $k \in Z$.

4. При четном N и $k \in 0: N/2 - 1$ справедливы равенства

$$X(2k) = \sum_{j=0}^{\frac{N}{2}-1} [x(j) + x(N/2 + j)] * \omega_{N/2}^{-kj},$$

$$X(2k+1) = \sum_{j=0}^{\frac{N}{2}-1} [x(j) - x(N/2 + j)] * \omega_N^{-j} * \omega_{N/2}^{-kj}.$$

5. Пусть сигнал x_l связан с исходным сигналом $x \epsilon C_N$ соотношением

$$x_l(j) = x(j + l), \qquad \text{где } l \in Z.$$

Тогда спектры X_l, X связаны соотношением

$$X_l(k) = \omega_N^{kl} * X(k), \qquad k \in Z.$$

6. Пусть сигнал x_l связан с исходным сигналом $x \in C_N$ соотношением

$$x_l(j) = \cos\frac{2\pi lj}{N} * x(j), \quad \text{где } l \epsilon Z.$$

Тогда для спектров X_l, X справедливо равенство

$$X_l(k) = \frac{1}{2}\big(X(k-l) + X(k+l)\big), \qquad k \in Z.$$

7. Сигнал x_n является удлинением сигнала $x \in C_N$, т. е.

$$x_n(j) = \begin{cases} x(j), & \text{при } j \in 0: N-1, \\ 0, & \text{при } j \in N: nN-1 \ \ (x_n \in C_{nN}). \end{cases}$$

Тогда их спектры X_n, X связаны формулой

$$X_n(k) = \sum_{l=0}^{N-1} X(l) * \bar{h}(k - ln), \qquad \text{где } h(k) = \frac{1}{N}\sum_{j=0}^{N-1} \omega_{nN}^{kj}.$$

8. Сигнал x_n является растяжением сигнала $x \in C_N$, т. е.

$$x_n(j) = \begin{cases} x(j/n), & \text{если } \langle j \rangle_n = 0, \\ 0, & \text{при остальных } j \in Z \ (x_n \in C_{nN}). \end{cases}$$

Тогда для их спектров X_n, X справедливо равенство

$$X_n(k) = X(\langle k \rangle_n).$$

9. Сигнал y_n является прореживанием сигнала $x \in C_N$, т. е.

$$y_n(j) = x(jm) \text{ при } N = mn \ \ (y_n \in C_n).$$

Тогда для их спектров Y_n, X справедливо равенство

$$Y_n(k) = \frac{1}{m}\sum_{p=0}^{m-1} X(k + pn).$$

Пусть $X = F_N(x)$. Тогда сигнал $|X|$ называется амплитудным спектром, а сигнал $|X|^2$— спектром мощности сигнала x [1].

1.3 Расстояние Кульбака-Лейблера

Расстоянием Кульбака-Лейблера называют функционал, определяющий меру отдаленности друг от друга распределений вероятностей. Пусть P_1 и P_2 непрерывные вероятностные распределения, тогда расстояние Кульбака-Лейблера между ними определяется

$$DKL(P_1, P_2) = \int p_1(x) * \log_a \frac{p_1(x)}{p_2(x)} dx$$

где $p_1(x)$ – вероятность появления x в P_1, $p_2(x)$ – вероятность появления x в P_2, a – основание логарифма, в котором рассчитывается расстояние, x – значение из области X.

Расстояние Кульбака-Лейблера не является симметричной мерой определения расстояние между распределениями, то есть

$$DKL(P_1, P_2) \neq DKL(P_2, P_1)$$

для произвольных распределений вероятностей P_1 и P_2.

Кроме того, расстояние Кульбака-Лейблера является неотрицательной величиной

$$DKL(P_1, P_2) \geq 0.$$

Такое соотношение более известно как неравенство Гиббса.

Чем больше значение расстояния Кульбака-Лейблера, тем менее схожи данные вероятностные распределения, соответственно, чем меньше значение меры, тем более схожи. Равенство нулю расстояния Кульбака-Лейблера свидетельствует о том, что рассматриваемые распределения равны

$$DKL(P_1, P_2) = 0 \leftrightarrow P_1 = P_2$$

Расстояние Кульбака-Лейблера для дискретных плотностей распределения определяется формулой

$$DKL(P_1, P_2) = \sum_x p_1(x) * \log_a \frac{p_1(x)}{p_2(x)} \qquad (1.3)$$

где P_1 и P_2 распределения вероятностей, $p_1(x)$ – вероятность появления x в P_1, $p_2(x)$ – вероятность появления x в P_2, a – основание логарифма, в котором рассчитывается расстояние [2]. Если i-ый член $p_1(x) = 0$, то вклад i-ого слагаемого в сумму считается равным 0, так как $\lim_{x \to 0} x * \log x = 0$.

2 Существующие решения для определения схожести музыкальных произведений

В данном разделе будет рассмотрен вопрос существующих и ранее созданных проектов по данной или схожей тематике. К сожалению, в открытом доступе оказалось не много содержательной информации по теме работы, однако некоторые примеры все-таки были найдены.

Далее будут рассмотрены найденные проекты, связанные с определением схожести музыкальных произведений.

2.1 Приложение «Shazam»

Приложение «Shazam» [3] производит распознавание музыкального произведения. Сутью работы приложения является определение названия проигрываемого музыкального произведения в реальном времени. Входом для данного приложения является музыкальная композиция, название которой пользователь хочет узнать, а выходом – собственно, название трека. Приложение находится в свободном доступе и очень широко распространено, имея 500+ млн. скачиваний в интернет-магазине «Google Play». Однако исходный код приложения является коммерческой тайной, и потому в свободном обращении его найти невозможно, собственно, в связи с этим узнать алгоритм определения схожести музыкального произведения, подаваемого на вход, с существующими в базе данных приложения «Shazam» слепками музыкальных треков, нельзя. Также стоит отметить, что приложение решает и несколько другую задачу в целом – вопрос идентификации любых музыкальных произведений, подаваемых на вход в реальном времени, в то время как цель данной работы – определить метрику схожести музыкальных произведений. Несмотря на то, что приложение является коммерческим, некоторая информация о принципах работы приложения была опубликована создателем приложения – Эвери Ли Чунь Вонг (Avery Li-Chung Wang), в которой автор рассказал, каким образом была разработана и внедрена в коммерческую

эксплуатацию гибкая поисковая система аудио, алгоритм которой устойчив к шумам и искажениям, вычислительно эффективный и масштабируемый, способный быстро идентифицировать короткий фрагмент музыки, записанный через микрофон мобильного телефона в присутствии голосов переднего плана и других доминирующих шумов, а также через сжатие голосового кодека базы данных, содержащей более миллиона треков [3].

2.2 Социальная сеть «ВКонтакте»

Социальная сеть «ВКонтакте» [4] поддерживает возможность прослушивания, загрузки и обмена аудиозаписями. Механизм определения схожести музыкальных треков реализован в данной социальной сети в контексте сокращения используемых объемов памяти из-за загрузки одинаковых музыкальных произведений. Кроме того, определение одинаковых (или очень похожих) аудиозаписей позволяет:

- избежать дублирования в поиске одного трека под разными названиями;
- предлагать прослушать любимую композицию в более высоком качестве;
- усовершенствовать механизм рекомендаций;
- добавлять обложки и текст ко всем вариантам песни;

Приложение является коммерческим и, следовательно, получить доступ к исходному коду для ознакомления с используемыми алгоритмами нет возможности. Однако общий подход к реализации алгоритма определения схожести музыкальных произведений был описан одним из разработчиков «ВКонтакте» в своей статье, где был разъяснен механизм формирования и использования акустических отпечатков, их сравнение и централизованное хранение в базе данных отпечатков [4]. Стоит также отметить, что в данной социальной сети решается проблема хранения и подбора схожих музыкаль-

ных произведений, в то время как целью данной работы является определение метрики схожести между музыкальными произведениями.

2.3 Вывод

В процессе поиска информации по теме было найдено несколько широко известных продуктов, а именно социальная сеть «ВКонтакте» и приложение «Shazam», исходные коды которых являются коммерческой тайной, а также решающих задачу поиска схожих музыкальных произведений в первом случае, а во втором – идентификация трека по его звучанию.

Таким образом, не было найдено проектов, решающих задачу определения метрики схожести между музыкальными произведениями, а также код которых мог бы обращаться в открытом доступе.

3 Описание алгоритма определения схожести музыкальных произведений

В данном разделе будет описан разработанный алгоритм определения метрики схожести музыкальных композиций.

3.1 Формат входных данных

На вход алгоритма подаются два музыкальных произведения, метрику схожести между которыми необходимо определить. Входные файлы могут быть представлены в следующих форматах данных:

- MP3;
- OGG;
- WAV.

Если на вход были поданы файлы с форматами данных MP3 или OGG, то они конвертируются в файл формата WAV, и именно с данными в таком формате далее происходит вычисление метрики. Кроме того, если музыкальное произведение представлено в многоканальном виде (2 и более), то такой трек приводится к одноканальному формату путем усреднения значений по всем каналам для каждого отсчета. Унификация формата входных данных позволяет проводить корректное сравнение музыкальных произведений, представленных строго определенным образом.

Входной музыкальный трек является дискретным сигналом (рисунок 3.1). Данный сигнал рассматривается как аудио-временной ряд, то есть как некая зависимость f(t), где t_i – i-ый отсчет времени (принимает дискретные значения), а $f(t_i)$ – амплитудное значение i-ого отсчета. Количество отсчетов в секунду называется частотой дискретизации или частотой семплирования (Гц, Hz). При выполнении работы частота дискретизации входных музыкальных произведений была принята равной 44.1 кГц.

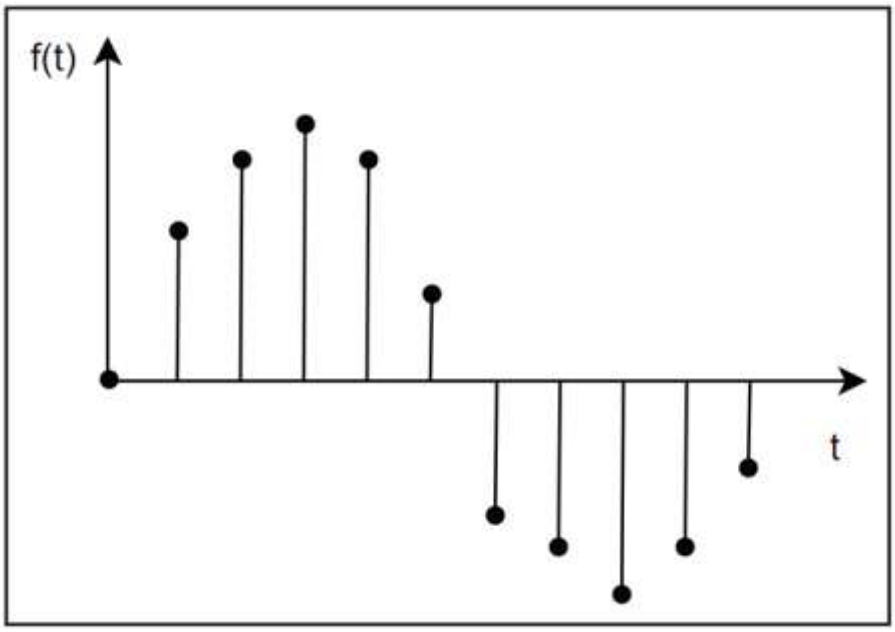

Рисунок 3.1 – Пример дискретного сигнала

Графически аудио-временной ряд можно представить в виде изображения огибающей амплитуды сигнала, которая отражает зависимость амплитуды от времени (рисунок 3.2).

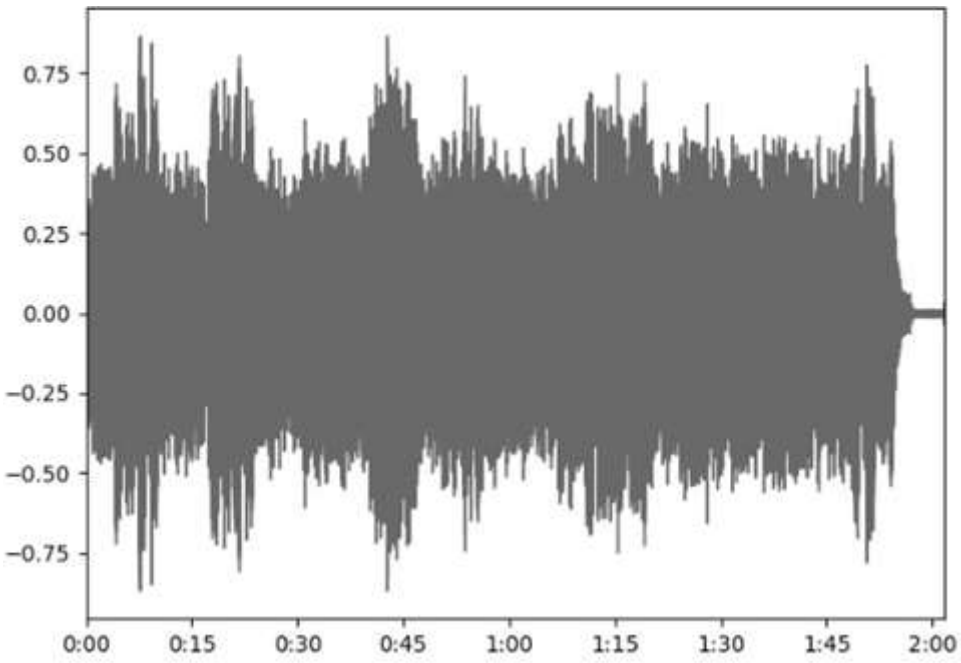

Рисунок 3.2 – Пример изображения огибающей амплитуды сигнала

3.2 Фильтрация частей музыкального трека

Рассматриваемое музыкальное произведение может состоять из разных частей с точки зрения восприятия - какой-то фрагмент может быть высокоинтенсивным и громким, какой-то быть тишиной, а какой-то и вовсе посторонним звуком – шумом. Для анализа на предмет схожести необходимо взять от сравниваемых треков самые содержательные части (самые энергетически сильные), которые являются для каждого из них характерными. Сравнение таких частей и будет определять значение метрики схожести между данными треками.

Чтобы произвести фильтрацию, необходимо разбить музыкальный трек на кадры (фреймы) по 8192 точки в каждом (рисунок 3.3). Таким образом, для дискретного сигнала длиной в N точек количество фреймов k будет равно

$$k = \begin{cases} \dfrac{N}{8192}, & \text{при } N \vdots 8192, \\ \left[\dfrac{N}{8192}\right] + 1, & \text{в остальных случаях.} \end{cases}$$

В случае если N не оказалось кратно 8192, то последний фрейм, дополняется нулями до размера 8192.

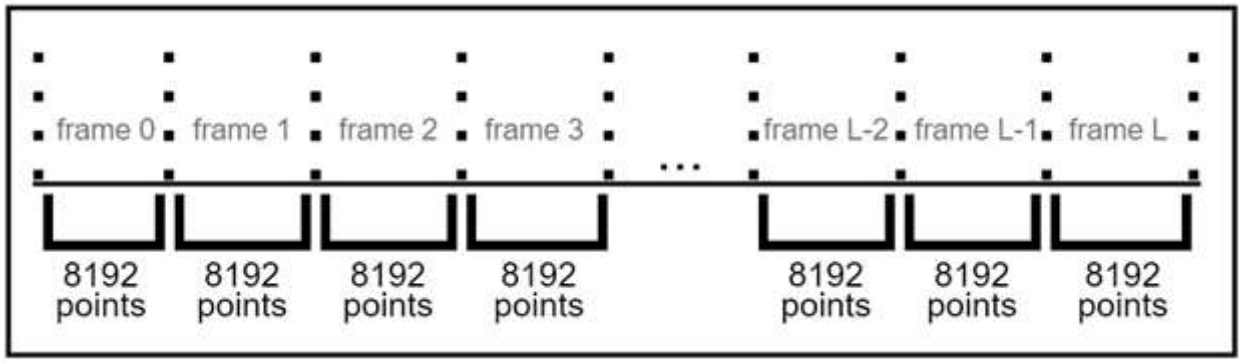

Рисунок 3.3 – Разбиение трека на кадры равной длины

Далее будут приведены подходы при фильтрации частей (кадров) музыкальных произведений.

3.2.1 Пороговая фильтрация

Суть данного подхода заключается в следующем: необходимо посчитать среднее модулей амплитудных значений для всего музыкального произведения $middle_{track}$, а также для каждого кадра (фрейма) в отдельности $middle_{frame}$. Далее для каждого фрейма нужно провести сравнение – если среднее модулей его амплитудных значений $middle_{frame}$ окажется больше или равно, чем среднее модулей амплитудных значений всего трека $middle_{track}$, умноженное на некоторый коэффициент k, то такой фрейм считается достаточно характерным для данного трека, чтобы его можно было использовать при дальнейшем сравнении, а если меньше, то данный фрейм отбрасывается и не будет использоваться в сравнении треков (рисунок 3.4):

$$\text{Если } middle_{frame} \geq k * middle_{track} \rightarrow \text{Фрейм подходит}$$

$$\text{Если } middle_{frame} < k * middle_{track} \rightarrow \text{Фрейм отбрасывается}$$

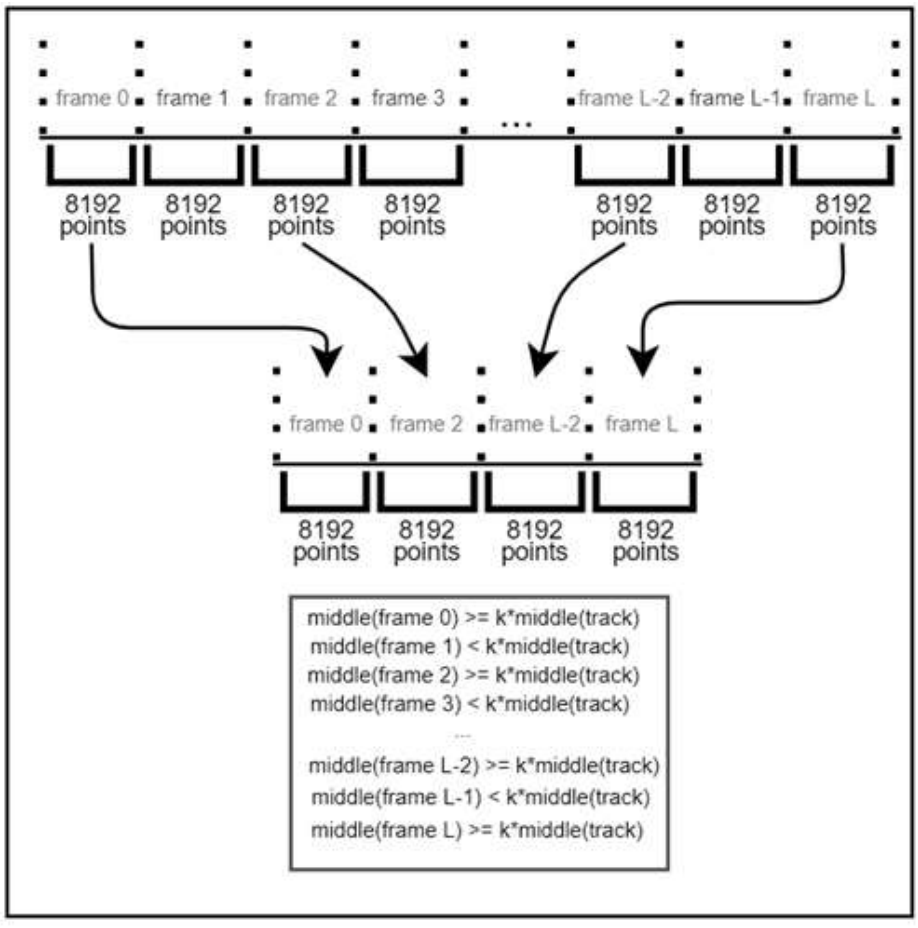

Рисунок 3.4 – Фильтрация трека по порогу

Коэффициент k подбирается экспериментально. В работе k был принят равным 0.25, так как такое значение позволяло отсеять не значимые фреймы, а также фреймы с шумами, при этом не отбрасывая содержательные части.

3.2.2 Фильтрация по числу фреймов

Данный способ фильтрации отбирает k самых значимых, сильных энергетически кадров музыкального произведения. Отбор происходит на основе среднего амплитудного значения кадров. Иными словами, при таком способе будут отобраны k кадров с наибольшими средними амплитудными значениями (рисунок 3.5).

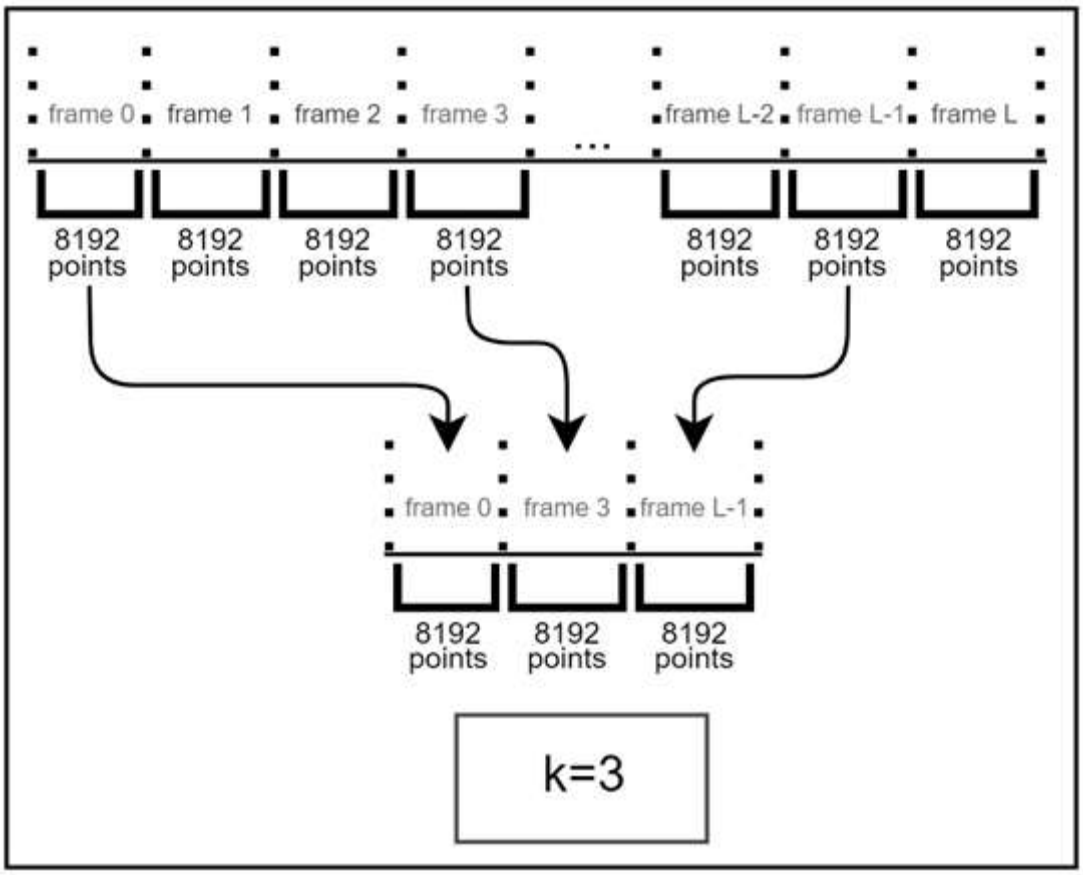

Рисунок 3.5 – Фильтрация трека по количеству фреймов

Параметр k подбирается экспериментально и может зависеть от общего числа фреймов в музыкальном произведении, ведь, логично предположить,

что чем дольше трек, тем больше частей (фреймов), которые его могут характеризовать и описывать уникальность.

3.2.3 Сравнение подходов к фильтрации

Пороговая фильтрация производит более слабую фильтрацию, отсеивая либо слишком незначительные части музыкального трека, либо части, в которых присутствуют шумы. Тем не менее, даже с учетом этого, количество рассматриваемых фреймов сокращается не сильно, и большая часть кадров трека все-таки попадает в совокупность, на основании которой будет производить измерение схожести. В связи с этим встает вопрос об избыточности рассматриваемой информации, которая неизбежно за собой влечет ряд негативных последствий, в том числе и увеличение времени работы алгоритма. С увеличением значения коэффициента увеличивается и шанс того, что значимая информация может быть отсеяна, и это напрямую повлечет за собой искажение значения метрики схожести треков. Помимо вышеперечисленного, при данном подходе число фреймов, которые будут отобраны для каждого из музыкальных произведений, заранее неизвестно, и, вероятно, будет различным, что также может оказывать влияние на последующее сравнение.

Фильтрация по числу фреймов является более жесткой фильтрацией, и менее гибкой, поскольку если при пороговой фильтрации число отобранных фреймов зависит от самого сигнала, то в данном случае, число фреймов, которые будут отобраны, жестко фиксировано, и не зависит от сигнала. Кроме того, при подобной жесткой фиксации отбираемых фреймов вероятна ситуация, при которой значимые кадры не буду отобраны для сравнения, что напрямую повлечет за собой изменение результата сравнения. Более того, при сравнении треков с ощутимо отличающейся длиной (и следовательно количеством кадров), одно фиксированное количество кадров для одного трека может включать большинство его кадров, а для другого лишь малую его

часть, что может также влиять на результат сравнения. В то же время жестко фиксированное количество отбираемых фреймов позволяет избежать переизбытка информации, что сократит время работы алгоритма и количество используемых ресурсов.

Таким образом, пороговая фильтрация является более предпочтительным вариантом обработки фреймов музыкальных произведений в виду гибкости и небольшой строгости при отборе.

3.3 Формирование нотного спектра

После фильтрации всех кадров музыкального произведения, был сформирован набор значимых кадров для данного трека. Именно с отфильтрованным набором кадров (ОНК) будет производиться работа далее. Каждый фрейм описывает некую зависимость амплитудных значений от времени в отсчетах, определяющих рассматриваемый фрейм. Взаимодействие с данными, представленными таким образом, не является удобным для последующего сравнения музыкальных произведений, поэтому для каждого кадра из ОНК выполним ДПФ согласно формулам 1.1 и 1.2, тогда результат ДПФ $X(k)$ каждого фрейма определяется по формуле

$$X(k) = \sum_{j=0}^{N-1} x(j) * \omega_N^{-kj}, \qquad \text{где } k \in 0: N-1,$$

$$\omega_N = \cos\frac{2\pi}{N} + i * \sin\frac{2\pi}{N} = e^{\frac{2\pi i}{N}}$$

Сигнал X называется спектром Фурье фрейма x или просто спектром, величины X(k) – компонентами спектра или спектральными составляющими, N=8192 – число отсчетов в кадре. С помощью ДПФ был произведен переход из временной области в частотную область, то есть теперь фреймы, характеризующие части музыкального трека, описываются не временем их наступления, а частотами. Стоит обратить внимание, что размерность фрейма после

ДПФ остается неизменной и равняется 8192 точкам. Кроме того, необходимо отметить, что исходный аудио-временной ряд состоял из вещественных амплитудных значений, поэтому согласно свойству ДПФ (пункт 1.2) спектр Фурье является четным.

Таким образом, после ДПФ каждый фрейм будет представлен дискретным набором значений в частотной области. Каждое значение является комплексным числом, характеризующим амплитуду и фазу для данной частоты. $k \in 0: N-1$ – индексы частот. Для получения значения частоты по индексу необходимо воспользоваться следующей зависимостью:

$$w(k) = k * \frac{s}{N},$$

где w(k) – частота, соответствующая индексу $k \in 0: N-1$, s=44100 – частота дискретизации, N=8192 – число точек в кадре. Также стоит заметить, что частотный набор для каждого фрейма одинаковый в силу единой для всего трека частоты дискретизации и одинакового числа отсчетов в каждом фрейме.

Согласно пункту 1.1 самая значимая информация, с точки зрения восприятия человеком, находится в диапазоне от 220 Гц до 4 кГц. Диапазон, включающий вышеуказанные частоты, будет рассматриваться при определении меры схожести музыкальных треков.

Далее необходимо произвести разбиение дискретного частотного спектра на ноты. Исходя из пункта 1.1, частоты нот необходимо рассматривать не линейным образом, а логарифмическим, что обусловлено физиологией человека. Сформируем 5 октав, в каждой из которой по 12 нот, таким образом, общее число нот будет равняться 60 (рисунок 3.6).

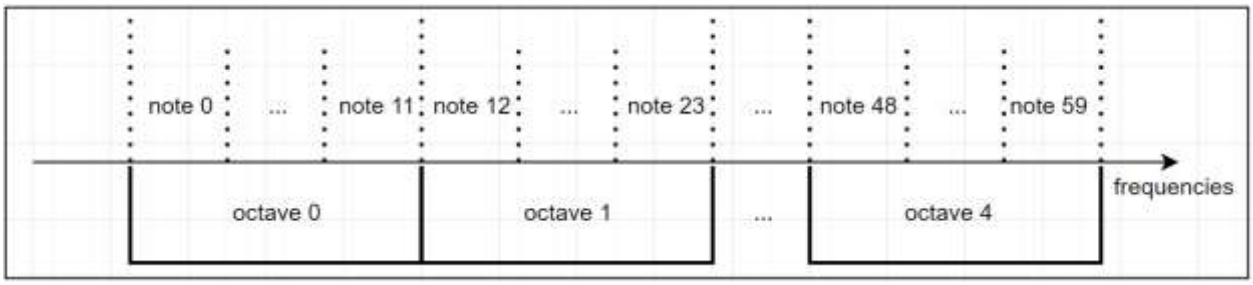

Рисунок 3.6 – Разбиение на ноты и октавы

Пусть опорной частотой нулевой ноты будет частота из дискретного набора w(k), наиболее близкая к 220 Гц, тогда опорной частотой первой ноты будет частота из w(k), наиболее близкая к $220*\sqrt[12]{2}$ Гц, второй – w(k), наиболее близкое к $220*(\sqrt[12]{2})^2$ Гц и так далее с шагом $\sqrt[12]{2}$. Границей нулевой ноты справа и первой слева будет частота из w(k), наиболее близкая к $\frac{220+220*\sqrt[12]{2}}{2}$ Гц, то есть частота из w(k), наиболее близкая к середине между 220 Гц и $220*\sqrt[12]{2}$ Гц (опорными частотами нулевой и первой нот соответственно). Разбиение происходит аналогичным образом для остальных нот (рисунок 3.7). Граница нулевой ноты слева определяется так, чтобы расстояние от опорной частоты до ее левой границы было максимально близким к расстоянию от опорной частоты до ее правой границы (правая граница 59-ой ноты определяется аналогичным образом).

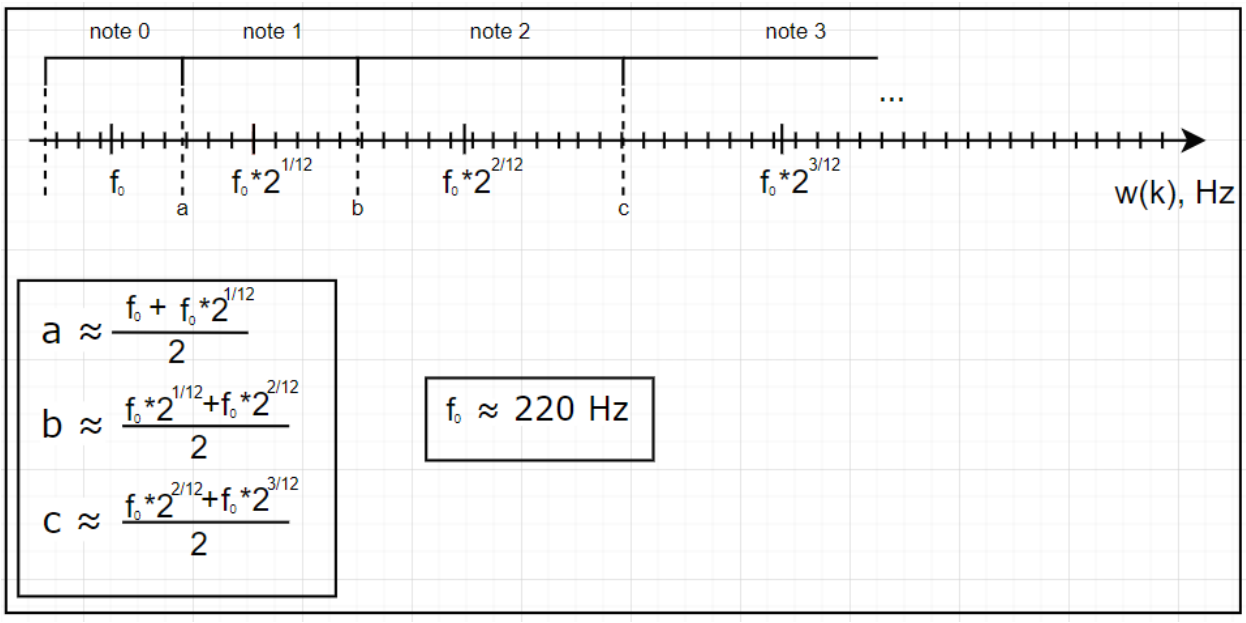

Рисунок 3.7 – Разбиение частот по нотам

При таком подходе к разбиению частот формируется логарифмический частотный спектр, где ширина частотного диапазона каждой следующей ноты увеличивается формально в $\sqrt[12]{2}$ раз по сравнению с шириной диапазона предыдущей ноты (на практике увеличение происходит в количество раз, близкое к $\sqrt[12]{2}$, поскольку опорные частоты и границы соответствующих нот выбираются из списка дискретных частот w(k), наиболее близким к ним), а ширина частотного диапазона каждой следующей октавы увеличивается формально в $\left(\sqrt[12]{2}\right)^{12} = 2$ раза по сравнению с шириной диапазона предыдущей октавы (на практике увеличение оказывается близко к ожидаемому значению, но не равно ему в силу, как уже выше было описано, дискретности значений частот w(k)). Таким образом, было произведено разбиение дискретного набора частот w(k) на 5 октав, каждая из которых содержит 12 нот, логарифмическим образом.

Далее необходимо сформировать матрицу нотного спектра (МНС). Подходы к формированию МНС описаны ниже.

3.3.1 Полный нотный спектр

После осуществления ДПФ для всех кадров из ОНК, было получено описание каждого фрейма в частотной области. Данное описание, как и во временной области, состоит из N=8192 точек для каждого кадра, значением каждой из которой является комплексное число, которое характеризует фазу и амплитуду для конкретной частоты, определяемой по индексу точки. Значение амплитуды определяется модулем комплексного числа, значение фазы – аргументом комплексного числа. Для получения полного нотного спектра произведем следующие действия:

1. Выберем один фрейм из ОНК после ДПФ.
2. Посчитаем среднее амплитудное значение всех частот для каждой из 60 сформированных ранее нот (частотных диапазонов) для выбранного фрейма.

3. Средние амплитудные значения каждой из нот для рассматриваемого фрейма поместим в вектор длиной 60, в порядке их вычисления, начиная с нулевой ноты и далее (рисунок 3.8).
4. Нормируем все значения вектора на максимальное значение в векторе. Таким образом, все элементы вектора должны лежать в диапазоне $[0; 1]$.
5. Выполним действия 2-4 для каждого из фреймов.

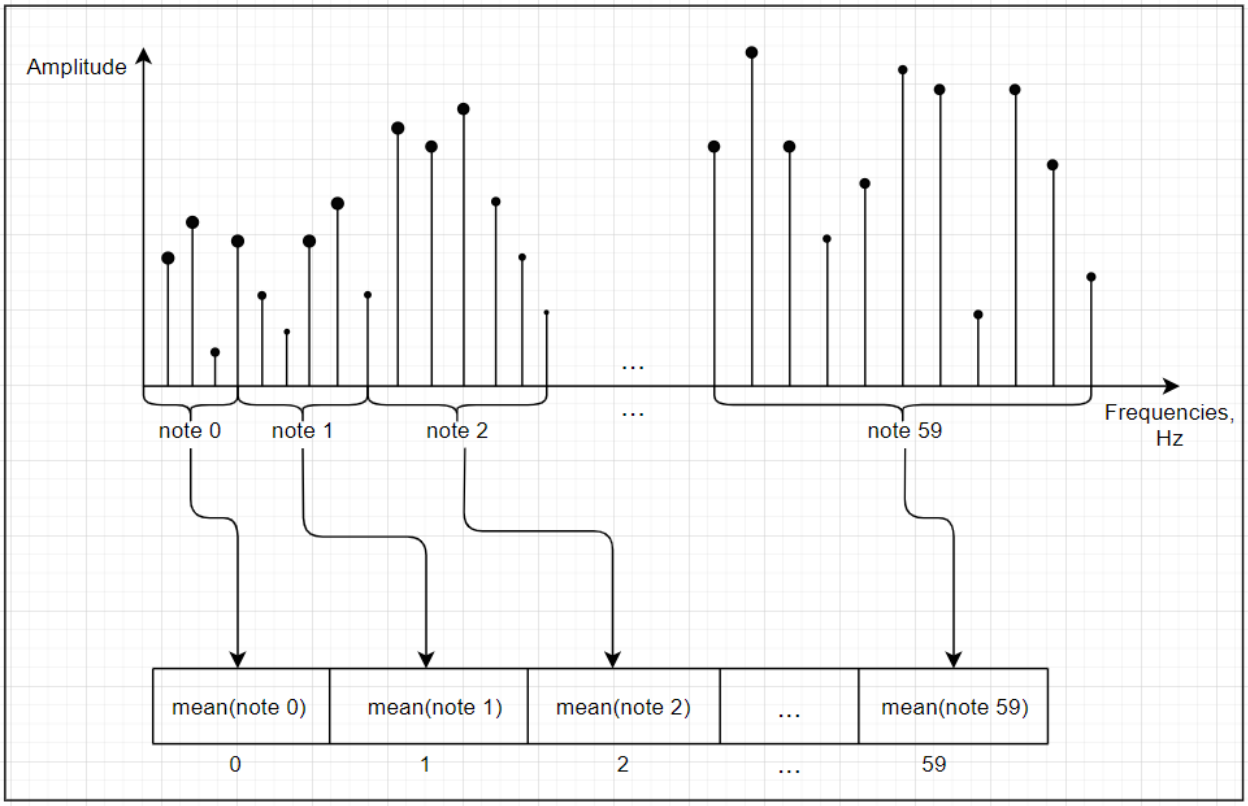

Рисунок 3.8 – Пример формирования вектора нотного спектра

Результат выполнения указанных выше операций для тестового фрейма представлен на графике (рисунок 3.9), который отражает зависимость среднего амплитудного значения от номера рассматриваемой ноты. По горизонтальной оси отложены номера нот, по вертикальной – нормированные значения среднего амплитудных значений для нот.

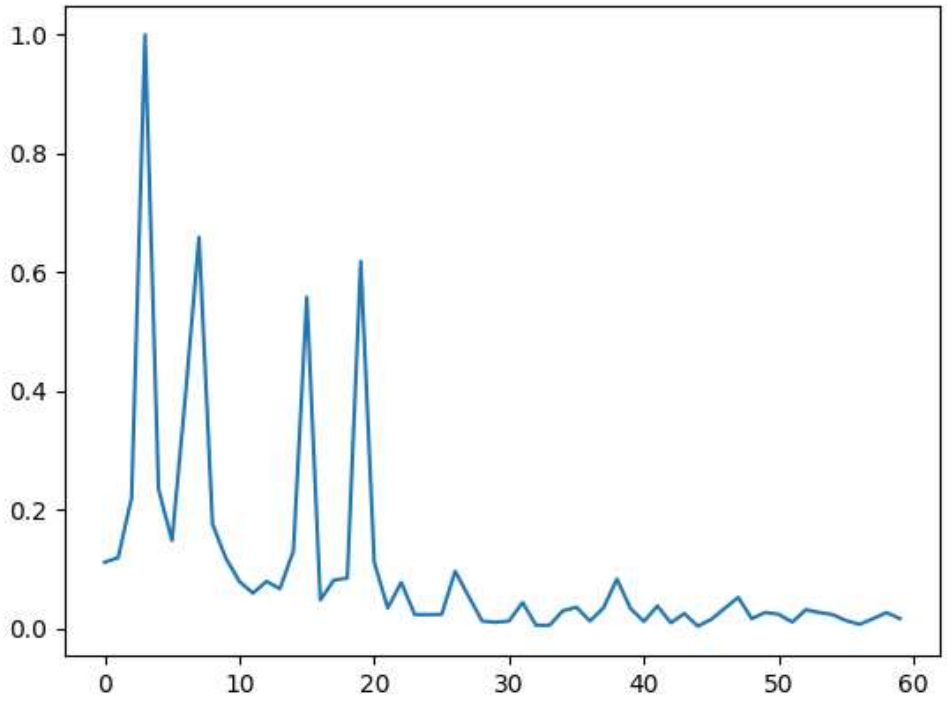

Рисунок 3.9 – Пример полного нотного спектра фрейма

Получив для каждого фрейма вектор длины 60, поместим эти вектора один за другим в матрицу построчно. Таким образом, мы сформируем матрицу нотного спектра (МНС) музыкального трека, в которой число строк равно числу фреймов в ОНК, а число столбцов равно числу нот – 60 (рисунок 3.10). В строке под номером i данной матрицы находится развертка средних амплитудных значений (нормированных к максимальному значению в данной строке) всех 60 нот для фрейма под номером i. В столбце под номером j находится развертка нормированных средних амплитудных значений всех фреймов для ноты (соответствующего диапазона частот) под номером j. На пересечении строки под номером i и столбца под номером j находится нормированное среднее амплитудное значение ноты под номером j фрейма под номером i.

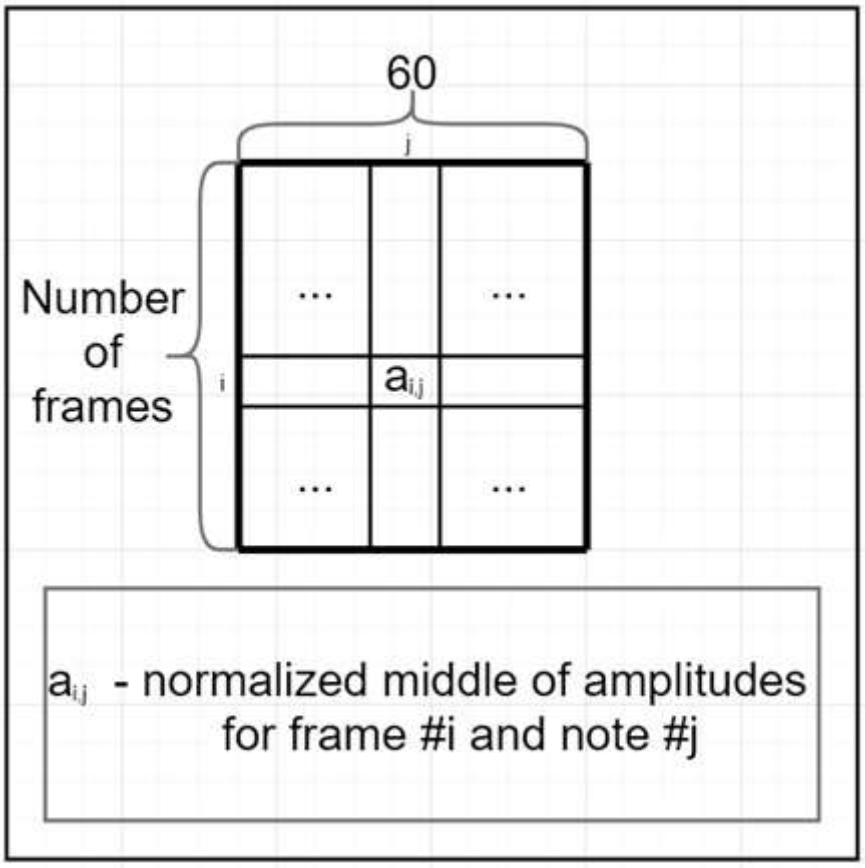

Рисунок 3.10 – Матрица полного нотного спектра

3.3.2 Метод транспонированной октавы

После осуществления ДПФ для всех кадров из ОНК, было получено описание каждого фрейма в частотной области. Данное описание, как и во временной области, состоит из N=8192 точек для каждого кадра, значением каждой из которой является комплексное число, которое характеризует фазу и амплитуду для конкретной частоты, определяемой по индексу точки. Значение амплитуды определяется модулем комплексного числа, значение фазы – аргументом комплексного числа. Метод транспонированной октавы для формирования МНС представлен ниже:

1. Выберем один фрейм из ОНК после ДПФ.

2. Посчитаем среднее амплитудное значение всех частот для каждой из 60 сформированных ранее нот (частотных диапазонов) для выбранного фрейма.
3. Найдем среднее от вычисленных средних амплитудных значений каждой 12 ноты, то есть найдем среднее от вычисленных значений для 0-й, 12-ой, 24-ой, 36-ой, 48-ой нот. Теперь найдем среднее для 1-й, 13-ой, 25-ой, 37-ой, 49-ой нот и так далее.
4. Средние значения, вычисленные на предыдущем шаге, поместим в вектор длиной 12, в порядке их вычисления, то есть под нулевым индексом в векторе будет находиться среднее для 0-й, 12-ой, 24-ой, 36-ой, 48-ой нот, под первым индексом – 1-й, 13-ой, 25-ой, 37-ой, 49-ой нот и так далее (рисунок 3.11).
5. Нормируем все значения вектора на максимальное значение в векторе. Таким образом, все элементы вектора должны лежать в диапазоне $[0; 1]$.
6. Выполним действия 2-5 для каждого из фреймов.

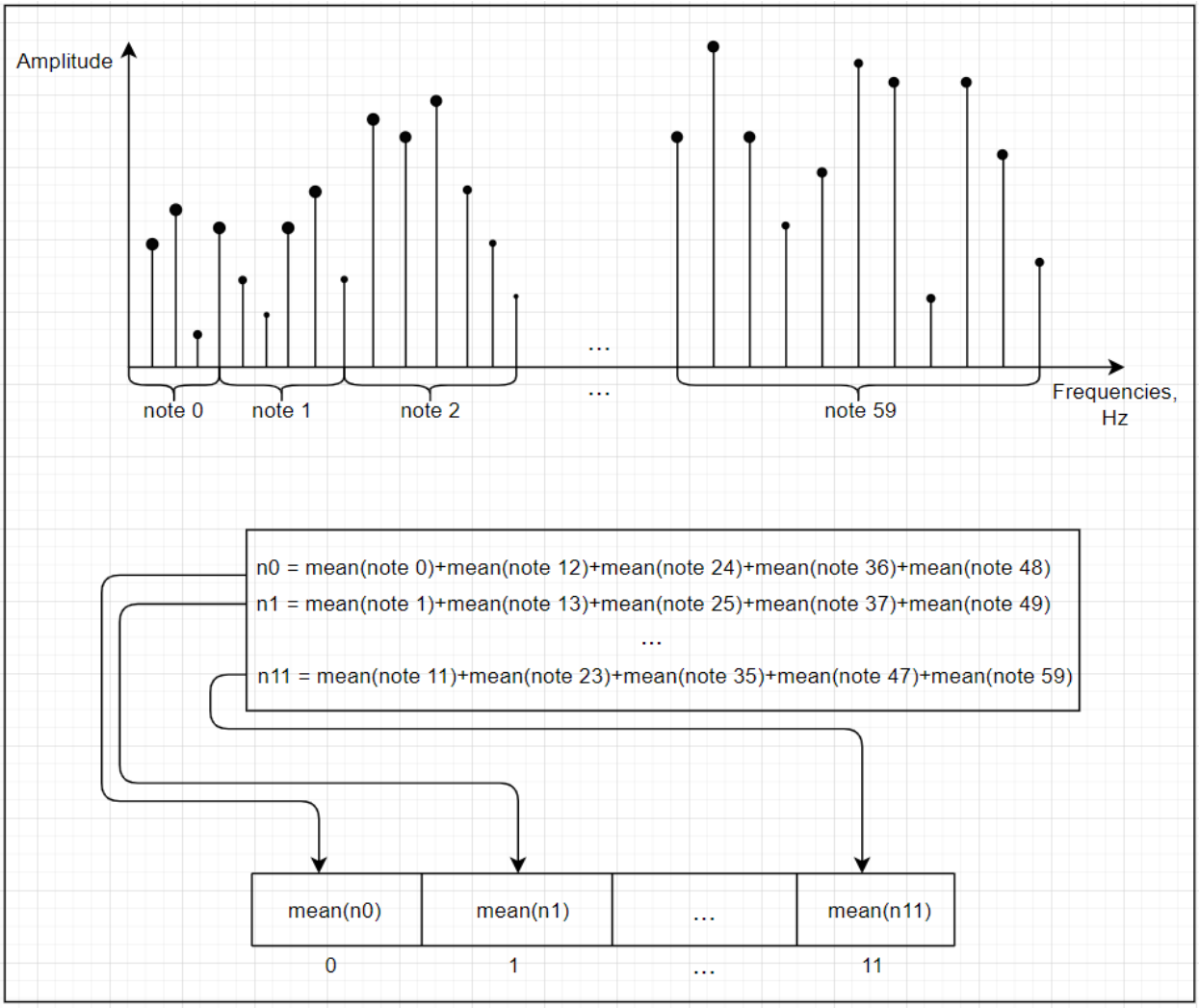

Рисунок 3.11 – Пример формирования вектора нотного спектра через транспонированную октаву

Результат выполнения указанных выше операций для тестового фрейма представлен на графике (рисунок 3.12), который отражает зависимость среднего амплитудного значения совокупности нот от номера рассматриваемой совокупности нот (всего их 12 штук). По горизонтальной оси отложены номера совокупностей нот, по вертикальной – нормированные значения среднего амплитудных значений для данных совокупностей нот.

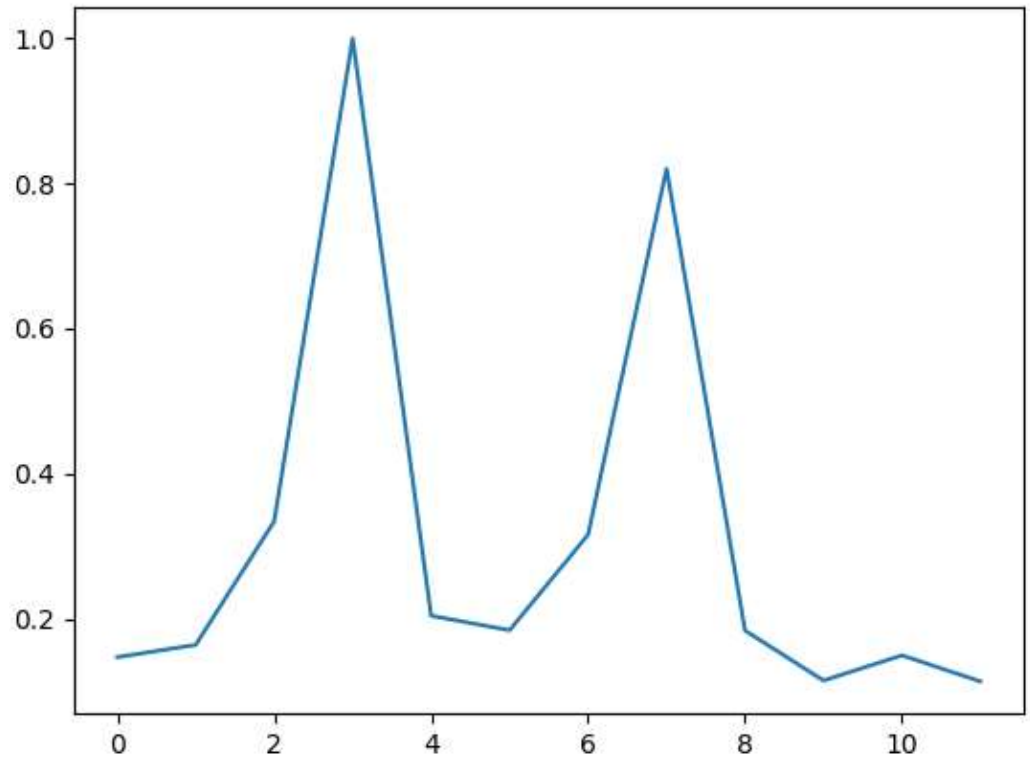

Рисунок 3.12 – Пример нотного спектра фрейма по методу транспонированной октавы

Получив для каждого фрейма вектор длины 12, поместим эти вектора один за другим в матрицу построчно. Таким образом, мы сформируем МНС музыкального трека, в которой число строк равно числу фреймов в ОНК, а число столбцов равно числу совокупностей нот после применения метода транспонированной октавы – 12 (рисунок 3.13). В строке под номером i данной матрицы находится развертка средних амплитудных значений (нормированных к максимальному значению в данной строке) 12 совокупностей нот для фрейма под номером i. В столбце под номером j находится развертка нормированных средних амплитудных значений всех фреймов для совокупности нот под номером j. На пересечении строки под номером i и столбца под номером j находится нормированное среднее амплитудное значение совокупности нот под номером j для фрейма под номером i.

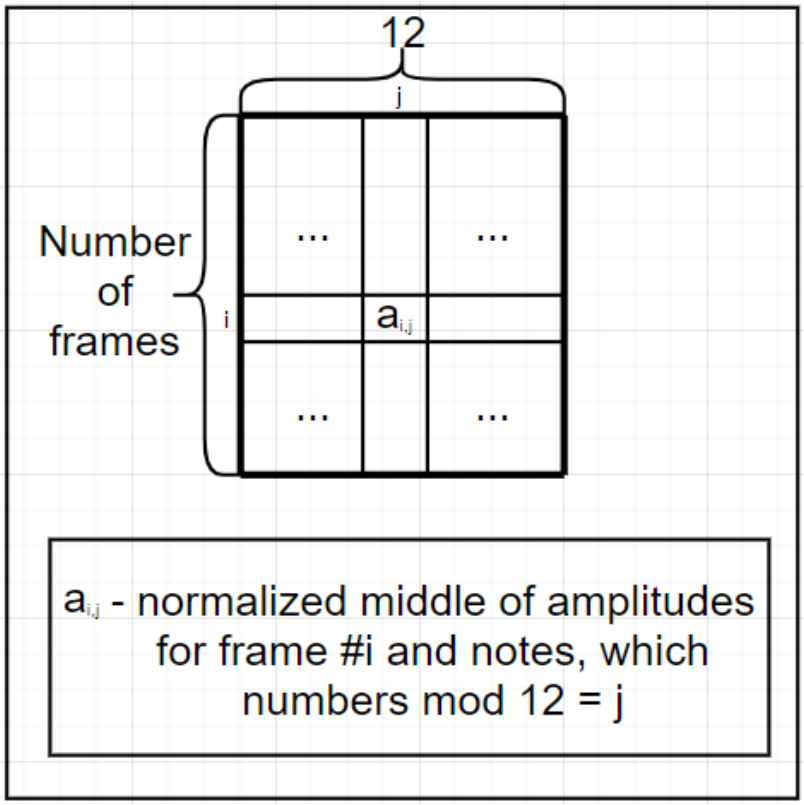

Рисунок 3.13 – Матрица нотного спектра по методу транспонированной октавы

3.3.3 Сравнение методов формирования МНС

Метод формирования полного нотного спектра описывает среднее амплитудное значение для каждой из 60 выделенных нот фрейма отдельно. Таким образом, сохранив все 60 значений (для каждой ноты), мы более полно и четко описываем характеристику фрейма на всем диапазоне частот, который интересен в рамках вычисления метрики схожести между музыкальными произведениями. Однако, стоит отметить, что из-за раздельного описания всех частотных диапазонов, данный метод не является устойчивым к каким-либо случайным звукам – шумам. Нота, содержащая шум, может иметь самое большое среднее из всех, и нормировка остальных амплитудных средних на такое самое большое среднее, очевидно, сильно исказит МНС, и как следствие, результат вычисления метрики между музыкальными произведениями.

Кроме того, хранение всех 60 средних амплитудных значений для каждого фрейма является не экономным вариантом работы с данными, которое влечет за собой большую трудоемкость алгоритма, и следовательно, увеличение времени обработки данных.

При использовании метода транспонированной октавы ноты разбиваются в 12 совокупностей по 5 нот, для каждой из таких совокупностей считается амплитудное среднее. Такой метод не описывает полно каждую ноту (каждый частотный диапазон), рассматриваемую для последующего вычисления метрики схожести между музыкальными произведениями. Однако, такой способ дает возможность уменьшить оказываемое влияние случайных звуков, шумов, помех, которые могли ошибочно пройти фильтрацию, на последующее сравнение треков, путем усреднения амплитудных значений нот по всем пяти октавам. Кроме того, такой способ является эффективным с точки зрения объема хранимых данных, поскольку для каждого фрейма хранятся лишь 12 значений, характеризующих его. Меньший объем хранимых данных влечет за собой уменьшение времени работы алгоритма.

Таким образом, более предпочтительным вариантом является метод транспонированной октавы, так как при его использовании влияние шумов на конечный результат меньше, чем при формировании МНС через полный нотный спектр. Также при использовании метода транспонированной октавы объем хранимых данных сокращается в 5 раз, так как вместо 60 столбцов хранится всего 12, а уменьшение объема данных увеличивает быстроту их обработки. Однако метод полного спектра более полно и детально описывает нотный спектр музыкального произведения.

3.4 Получение плотности распределения вероятностей

После формирования МНС для музыкальных произведений, метрику схожести между которыми необходимо проверить, будут получены, в общем случае, матрицы разных размеров (в силу разного количества рассматриваемых фреймов у каждого трека), что усложняет процесс сверки полученных нотных спектров. Стоит заметить, что все элементы МНС принадлежат [0;1], так как на последнем шаге формирования матрицы происходит нормировка каждой ее строки к максимальному значению в данной строке. В связи с этим сформируем матрицу плотности распределения вероятностей (МПРВ) для нот (или совокупности нот) и соответствующих им средних амплитудных значений по следующему принципу:

1. Произведем разбиение интервала возможных значений элементов МНС [0;1] на 10 интервалов с шагом 0.1, то есть на интервалы [0;0.1), [0.1;0.2), …, [0.8;0.9), [0.9;1].
2. Для каждого столбца МНС подсчитаем количество значений в данном столбце, которые попадают в каждый из 10 выделенных интервалов (рисунок 3.14). Для каждого столбца запишем результат в вектор длины 10.
3. Для каждого полученного вектора произведем нормирование на число, равное количеству фреймов (количеству строк в МНС). Таким образом, значения в векторе будут находиться в диапазоне [0;1].

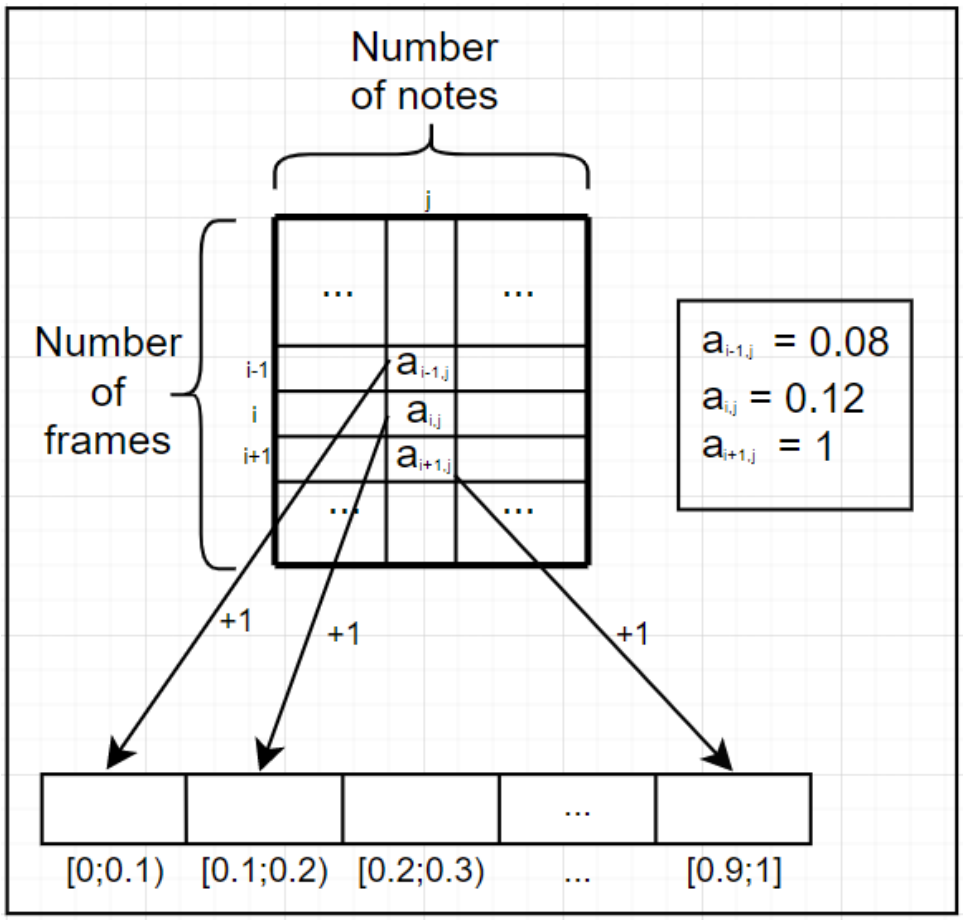

Рисунок 3.14 – Пример формирования плотности распределения вероятностей

Результат выполнения указанных выше операций для тестового фрейма представлен на графике плотности распределения вероятностей для данного фрейма (рисунок 3.15), который отражает зависимость значения вероятности попадания среднего амплитудного значения в соответствующий числовой диапазон. По горизонтальной оси отложены диапазоны значений, по вертикальной – значения вероятностей для соответствующих числовых диапазонов.

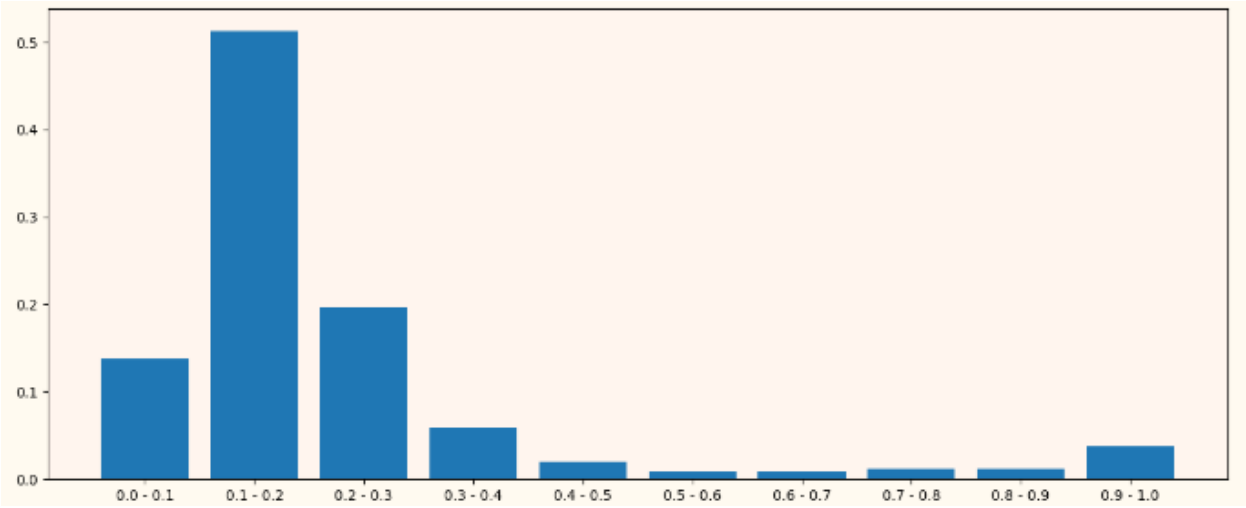

Рисунок 3.15 – Пример плотности распределения вероятностей для тестового фрейма

Полученные вектора (их количество равняется числу столбцов в МНС - числу рассматриваемых нот или совокупностей нот) поместим один за другим в порядке их вычисления (вычисление производиться с нулевого столбца МНС и далее) в матрицу. Данная матрица будет содержать плотность распределения вероятностей (ПРВ) для рассматриваемых нот музыкального произведения и называться матрицей плотности распределения вероятностей (МПРВ). Количество строк данной матрицы будет равняться количеству столбцов в МНС (количеству рассматриваемых нот или совокупностей нот), количество столбцов будет равно количеству интервалов разбиения – 10 (рисунок 3.16). В строке под номером i будут находиться вероятности распределения средних амплитудных значений на 10 интервалов для ноты (совокупности нот) под номером i. Столбец под номером j содержит информацию о значениях вероятности каждой ноты (совокупности нот) для числового диапазона под номером j. На пересечении строки под номером i и столбца под номером j находится вероятность попадания в интервал под номером j среднего амплитудного значения ноты (совокупности нот) под номером i.

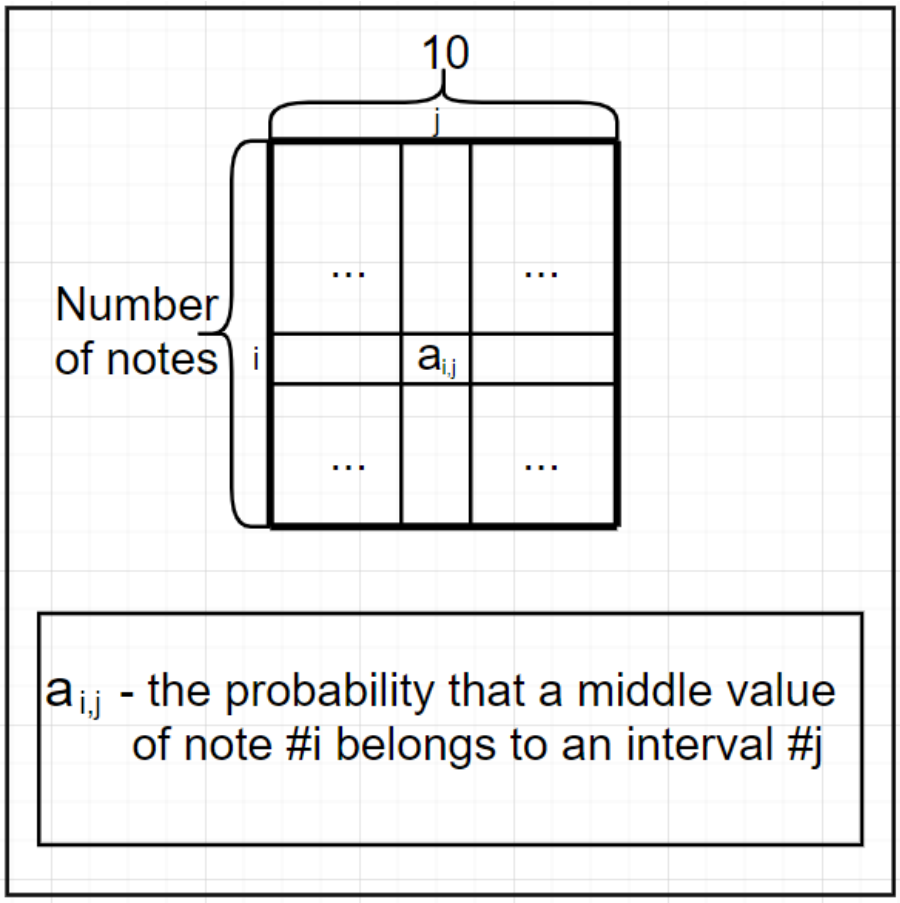

Рисунок 3.16 – Матрица плотности распределения вероятностей

3.5 Вычисление метрики схожести

По окончании процесса формирования МПРВ для музыкального произведения трек будет описываться исключительно с помощью данной матрицы. Важно отметить, что для всех музыкальных произведений вне зависимости от их длительности, характера звучания и так далее, такая матрица будет иметь одинаковые размеры, так как количество интервалов, на которые происходило разбиение значений, для всех едино и равно 10 – количество столбцов МПРВ. Количество строк матрицы так же является для всех единым, так как число рассматриваемых нот (совокупностей нот) выбирается для всех музыкальных произведений одинаковым и равняется либо 12, либо 60 (в зависимости от выбранного метода формирования нотного спектра, однако для сравниваемых музыкальных произведений метод должен быть оди-

наковым). Таким образом, единые размеры МПРВ, которые характеризуют треки, позволяют сравнить музыкальные произведения путем сравнения их МПРВ. Соответственно, для определения метрики схожести между треками необходимо определить метрику схожести между МПРВ. Метрикой, определяющей меру отдаленности распределений вероятностей, является расстояние Кульбака-Лейблера, которая определяется для дискретных величин согласно формуле 1.3.

Так как данные, описывающие музыкальные произведения представлены в виде матрицы, то вычислим метрику Кульбака-Лейблера последовательно для каждой из строк матриц попарно (рисунок 3.17).

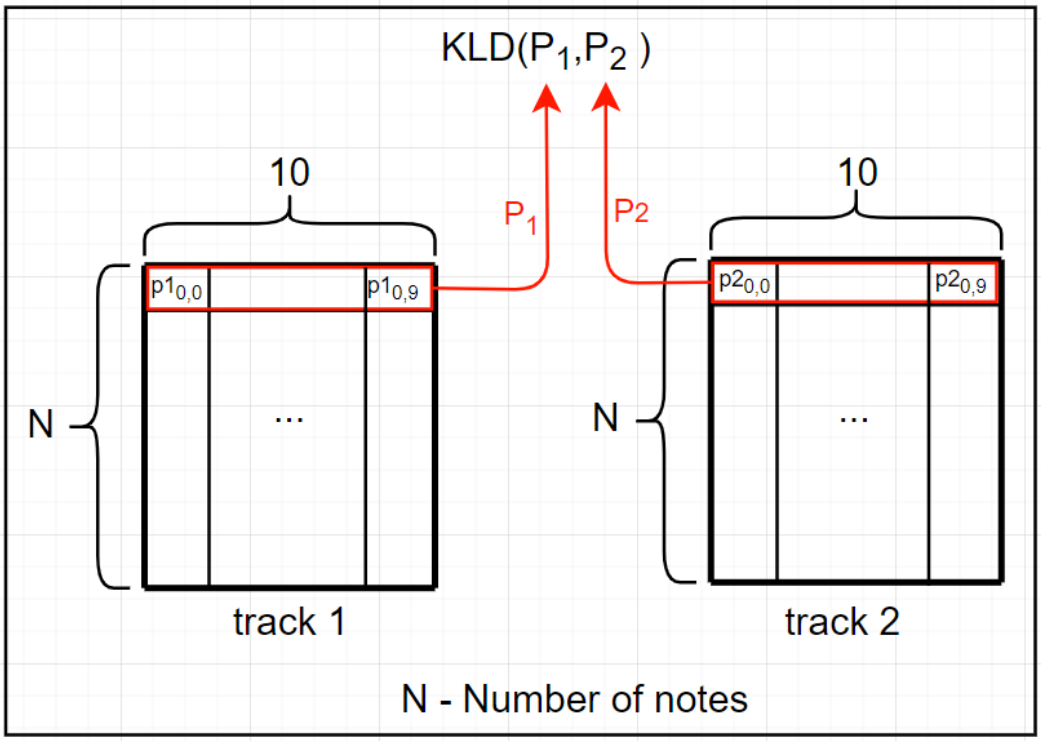

Рисунок 3.17 – Вычисление отдаленности распределений по строке

Вычислив метрику для пары нулевых строк матриц, для пары первых строк матриц и так далее, запишем полученные N значений (N – число строк в матрицах или же число рассматриваемых нот) последовательно в вектор соответствующей длины. Данный вектор содержит значения отдаленности распределений для каждой из строк матриц. Чем больше полученное значение для пары конкретных строк, тем менее они схожи. Таким образом, схожесть музыкальных произведений определяется полученным вектором. Ре-

зультат вычисления метрики для МПРВ двух тестовых музыкальных произведений представлен ниже (рисунок 3.18). Важно заметить, что метрика Кульбака-Лейблера является несимметричной, то есть $KLD(P_1, P_2)$ и $KLD(P_2, P_1)$, в общем случае, различны. Для тех же тестовых музыкальных произведений результат вычисления метрики с учетом замены местами P_1 и P_2 представлен ниже (рисунок 3.19).

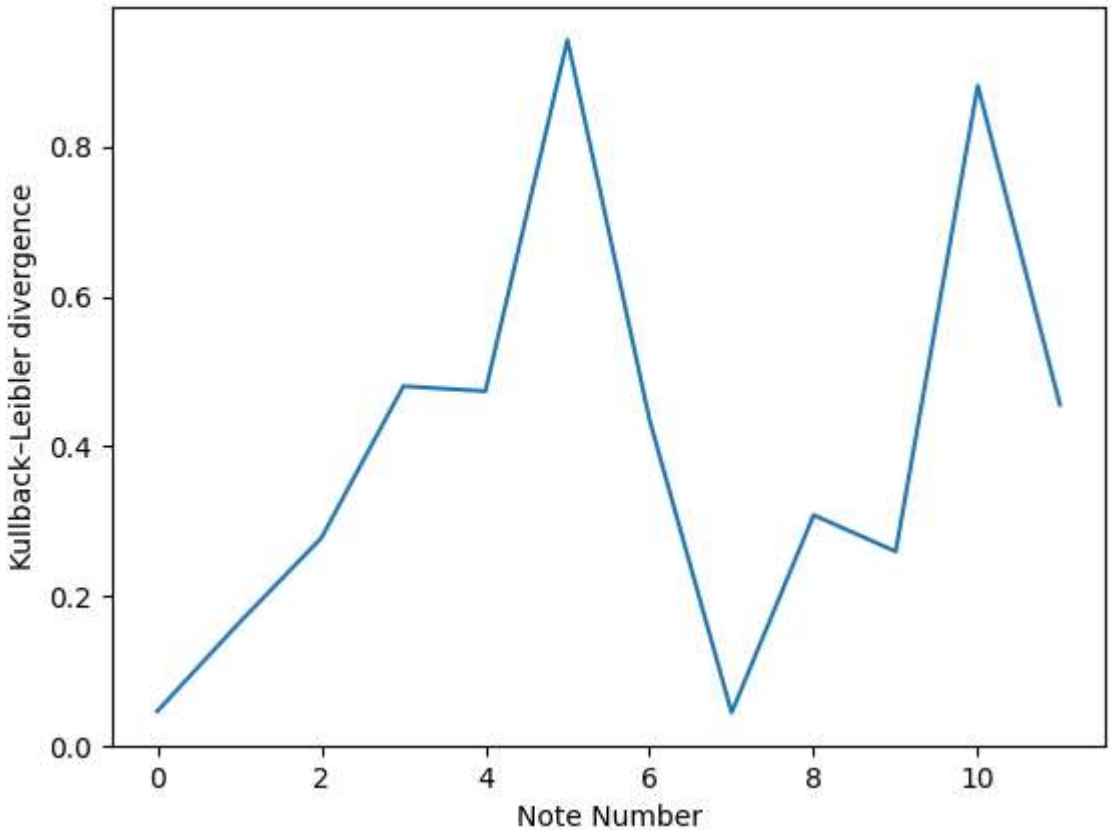

Рисунок 3.18 – Результат вычисления метрики Кульбака-Лейблера $KLD(P_1, P_2)$ для тестовых произведений

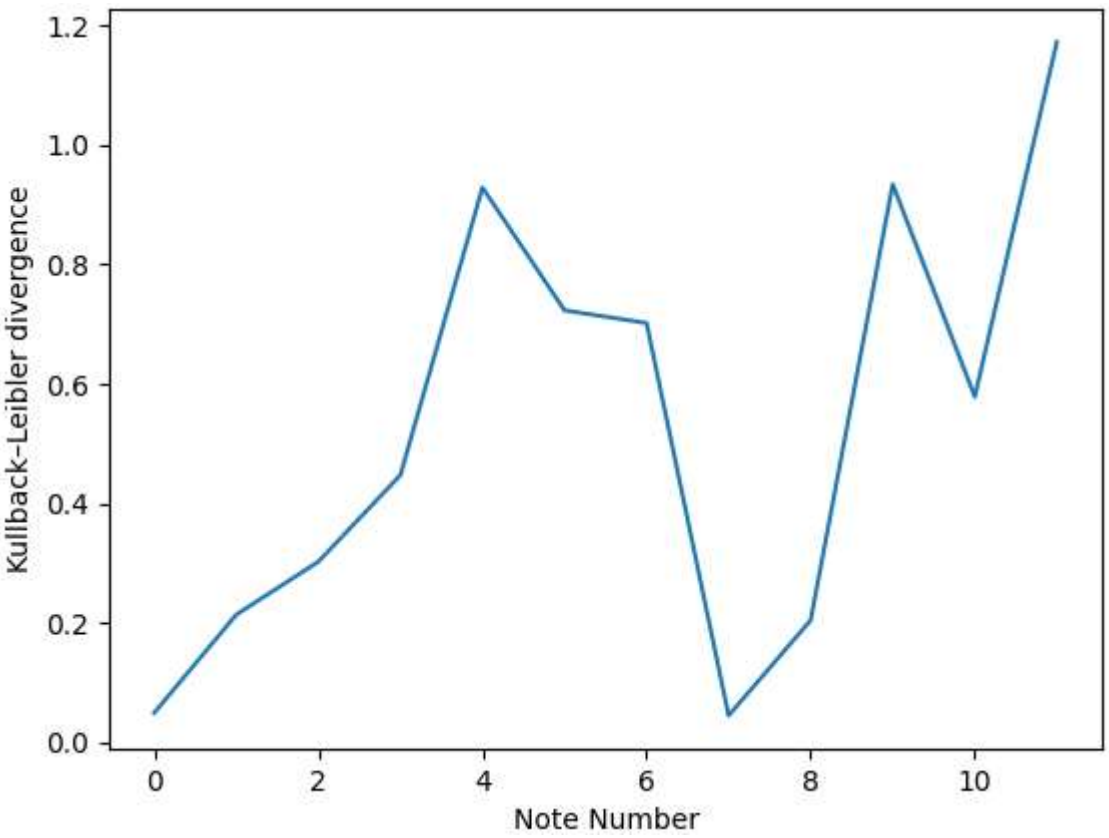

Рисунок 3.19 – Результат вычисления метрики Кульбака-Лейблера $KLD(P_2, P_1)$ для тестовых произведений

По горизонтальной оси отложены номера нот (совокупности нот), по вертикальной – расстояние Кульбака-Лейблера между соответствующими значениями нот двух музыкальных произведений.

Таким образом, для двух музыкальных произведений, в результате вычисления расстояния Кулбака-Лейблера между МПРВ, получается два вектора (в силу несимметричности используемой меры), которые описывают схожесть двух музыкальных произведений. Сформируем из двух векторов один, имеющий тот же размер, что и два предыдущих, и состоящий из средних значений элементов соответствующих позиций двух векторов. Результат выполнения данной операции представлен в виде графика вектора (рисунок 3.20) для двух ранее указанных тестовых музыкальных треков.

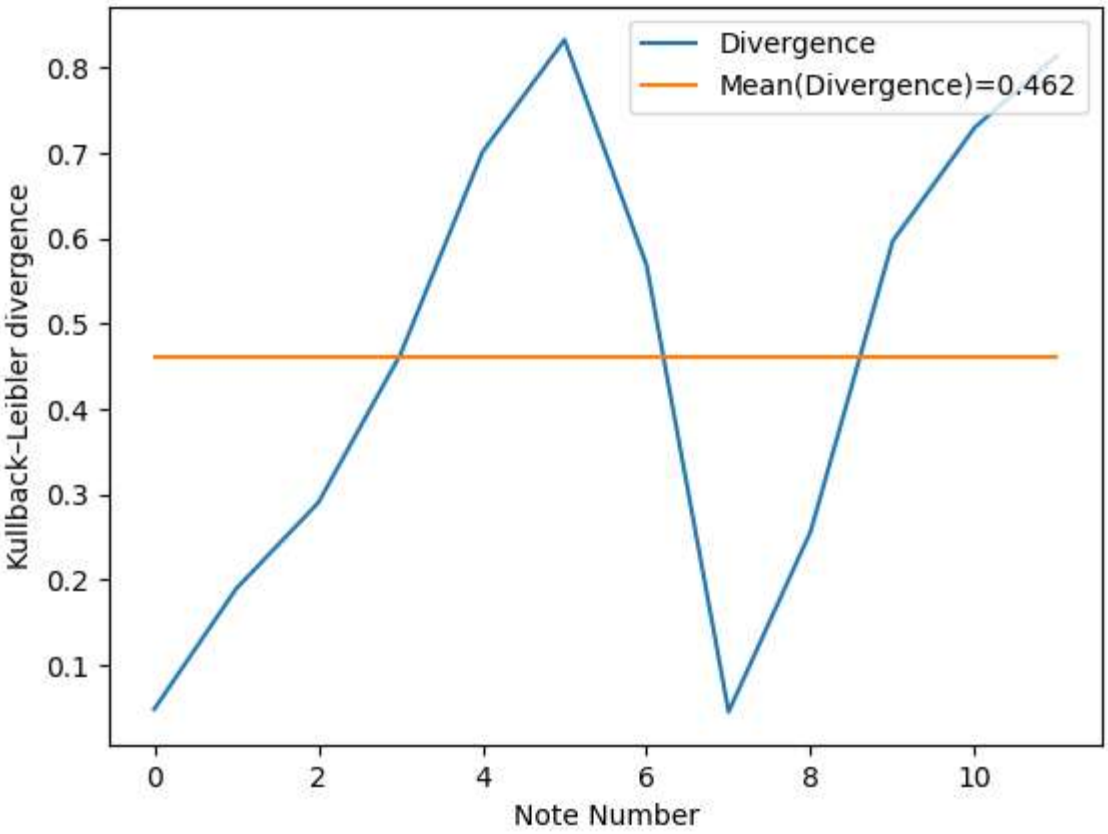

Рисунок 3.20 – Усредненное значение метрики Кульбака-Лейблера для тестовых произведений

Среднее значение получившегося вектора есть число, отражающее значение метрики схожести между двумя музыкальными произведениями. Чем выше значение данной метрики, тем более различны музыкальные произведения. Чем меньше значение метрики, тем более схожи музыкальные произведения.

4 Экспериментальная проверка разработанного алгоритма определения схожести музыкальных произведений

В данном разделе будет представлено тестирование программно реализованных вариантов алгоритма на языке Python, структуры и оптимальные варианты построения которых были изучены в специализированной литературе по данному языку программирования [7] [8] [9].

4.1 Пороговая фильтрация и метод транспонированной октавы

Проведем тестирование работы алгоритма, который использует фильтрацию исходных кадров музыкального трека по коэффициенту (по порогу) 0.25, а также для формирования нотного спектра использует метод транспонированной октавы.

Составим набор из тестовых музыкальных композиций Моцарта, Чайковского, Баха, Вивальди. Данный набор можно разбить на 4 условные группы:

1. Музыкальное произведение в 3 разных инструментальных исполнениях – итого 3 музыкальные композиции.
2. Музыкальное произведение (отлично от музыкального произведения в первом пункте) в 2 разных инструментальных исполнениях – итого 2 музыкальные композиции.
3. Музыкальное произведение (отлично от музыкальных произведений в первом и втором пунктах) в 2 разных инструментальных исполнениях – итого 2 музыкальные композиции.
4. Музыкальные произведения, отличные от произведений в 1-3 пунктах, а также отличные между собой – итого 2 музыкальные композиции.

Приведем одну из таблиц для анализа схожести между 1 произведением (Антонио Вивальди – Весна, «Аллегро», пианино) и остальными 8 (таблица 4.1).

Таблица 4.1.1 – Меры схожести между произведениями

Название	Схожи	Среднее значение схожести	Медианное значение схожести
BWV 565 Toccata Fugue in D Minor (harp)	Нет	0.708	0.519
BWV 565 Toccata Fugue in D Minor (organ)	Нет	0.908	0.822
BWV 565 Toccata Fugue in D Minor (piano)	Нет	0.776	0.727
Mozart – Maurerische Trauermusik	Нет	0.907	0.71
BWV Prelude 849 by Martin Stadtfeld	Нет	0.373	0.251
Chaykovskiy – Dance of the Sugar Plum Fairy (piano)	Нет	0.341	0.244
Chaykovskiy – Dance of the Sugar Plum Fairy (by The Big Theatre)	Нет	0.403	0.275
Vivaldi – Spring Allegro (violin)	Да	0.095	0.099

Значения в последних двух столбцах есть среднее и медианное значения после формирования вектора из средних от двух векторов – результатов расхождения Кульбака-Лейблера между двумя треками. Далее приведем графики векторов из средних для каждой пары из таблицы 4.1 (рисунки 4.1-4.8).

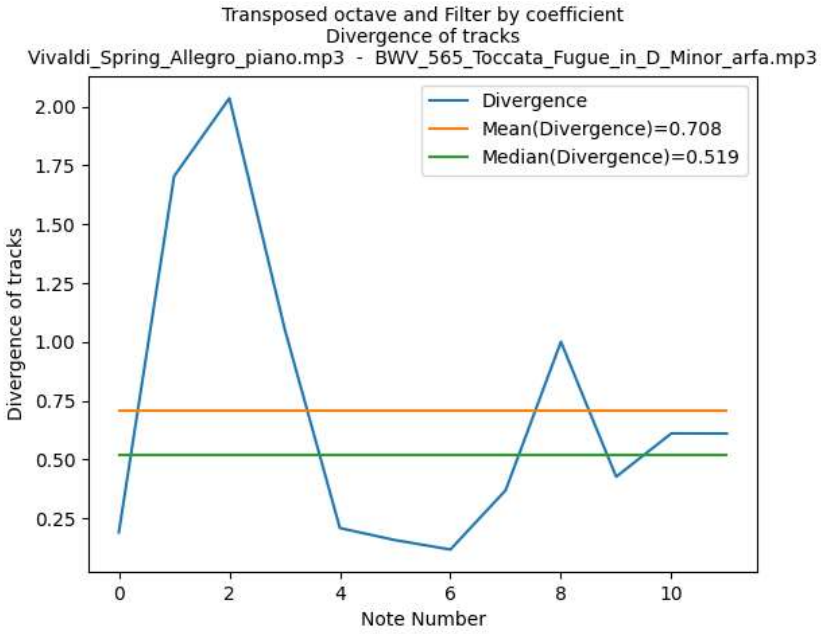

Рисунок 4.1 – Мера схожести между Vivaldi – Spring, Allegro (piano) и BWV 565 Toccata Fugue in D Minor (harp)

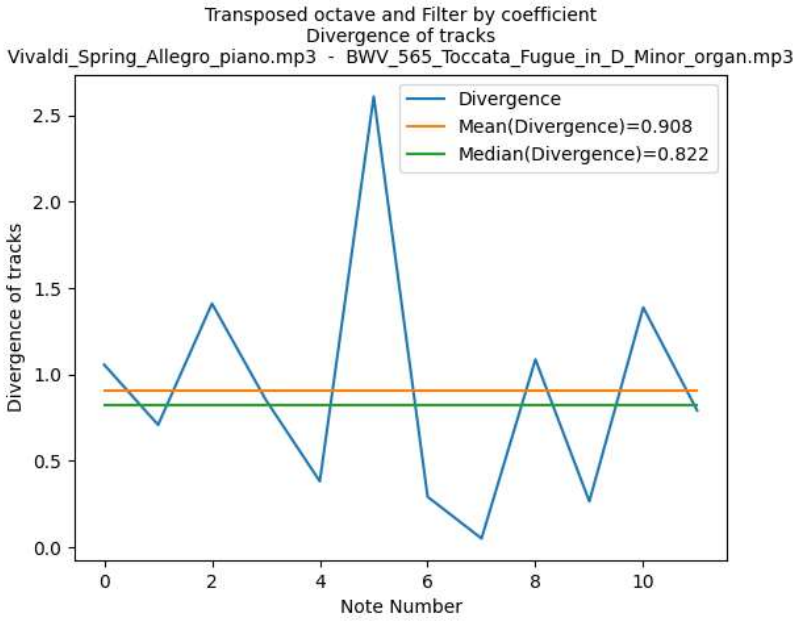

Рисунок 4.2 – Мера схожести между Vivaldi – Spring, Allegro (piano) и BWV 565 Toccata Fugue in D Minor (organ)

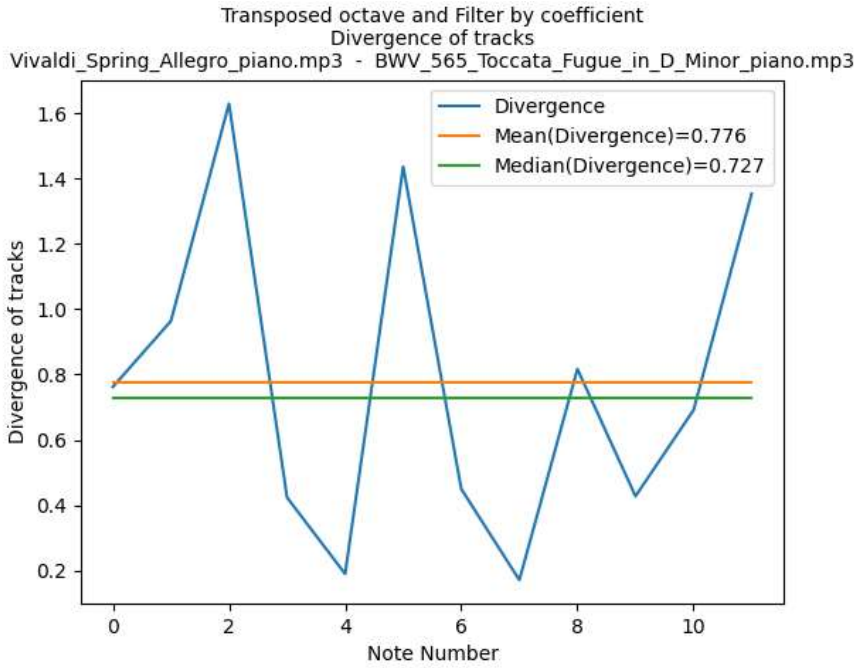

Рисунок 4.3 – Мера схожести между Vivaldi – Spring, Allegro (piano) и BWV 565 Toccata Fugue in D Minor (piano)

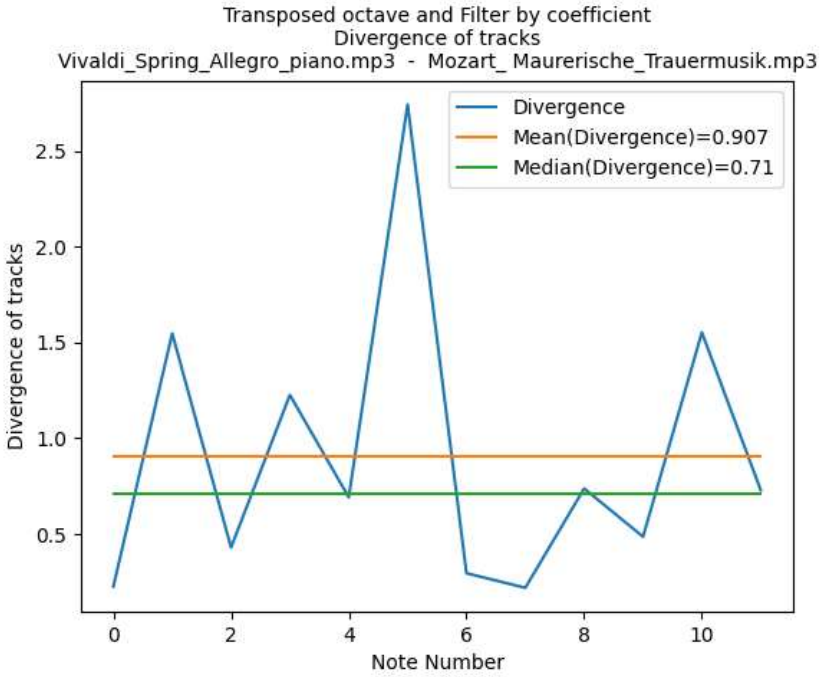

Рисунок 4.4 – Мера схожести между Vivaldi – Spring, Allegro (piano) и Mozart – Maurerische Trauermusik

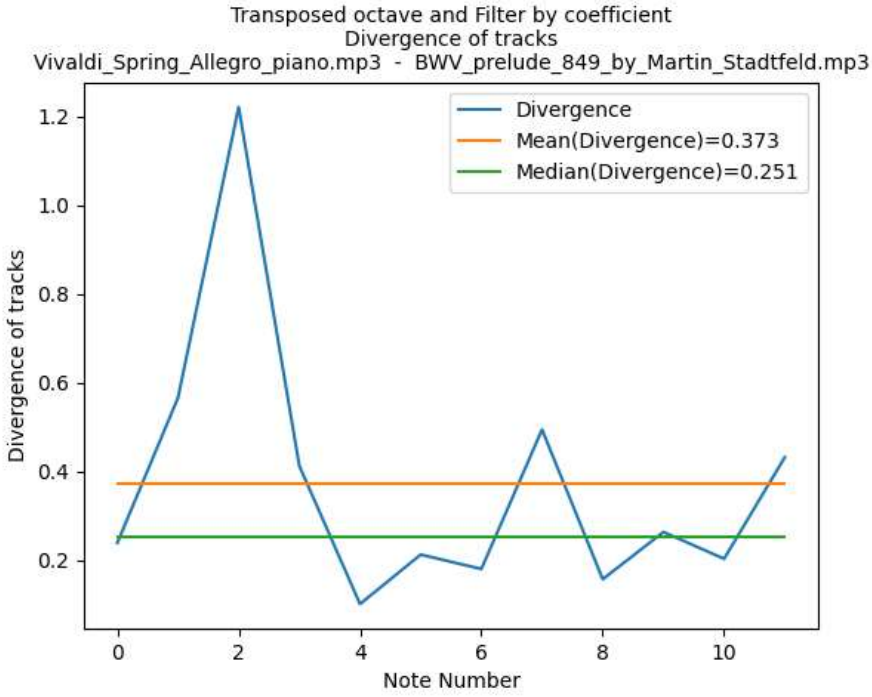

Рисунок 4.5 – Мера схожести между Vivaldi – Spring, Allegro (piano) и BWV Prelude 849 by Martin Stadtfeld

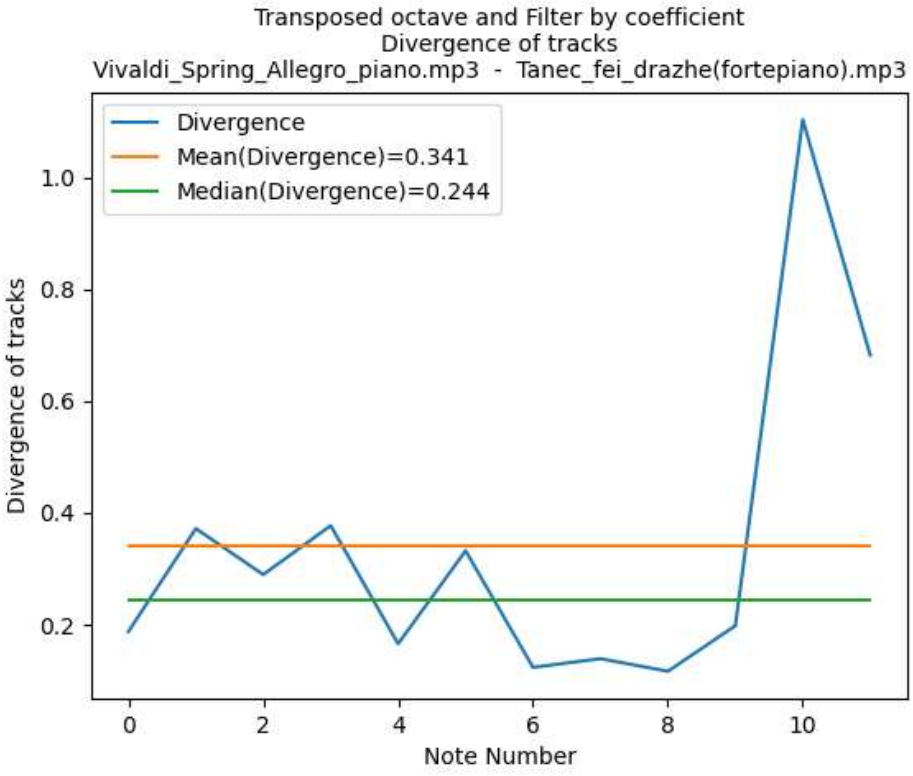

Рисунок 4.6 – Мера схожести между Vivaldi – Spring, Allegro (piano) и Chaykovskiy – Dance of the Sugar Plum Fairy (piano)

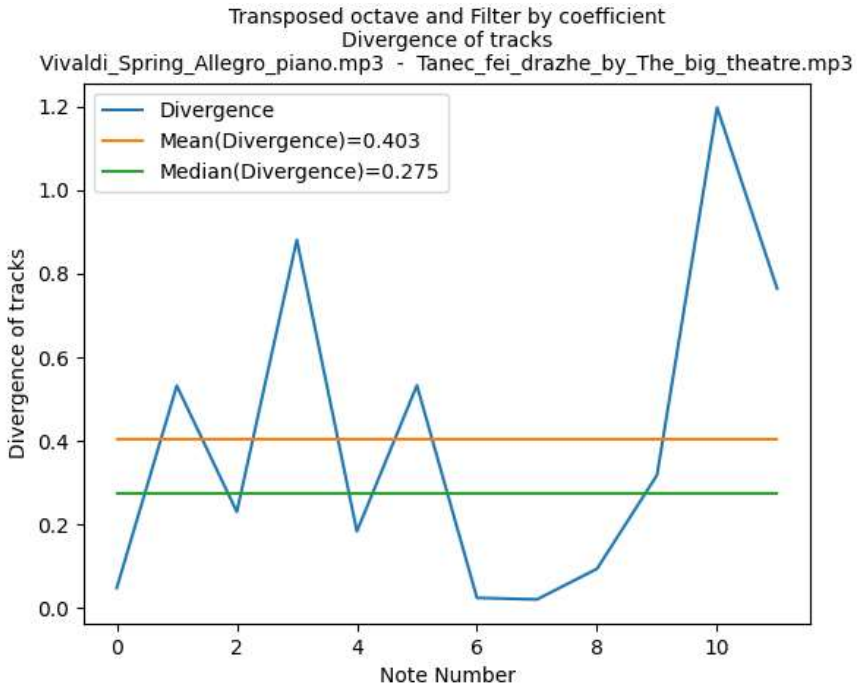

Рисунок 4.7 – Мера схожести между Vivaldi – Spring, Allegro (piano) и Chaykovskiy – Dance of the Sugar Plum Fairy (by The Big Theatre)

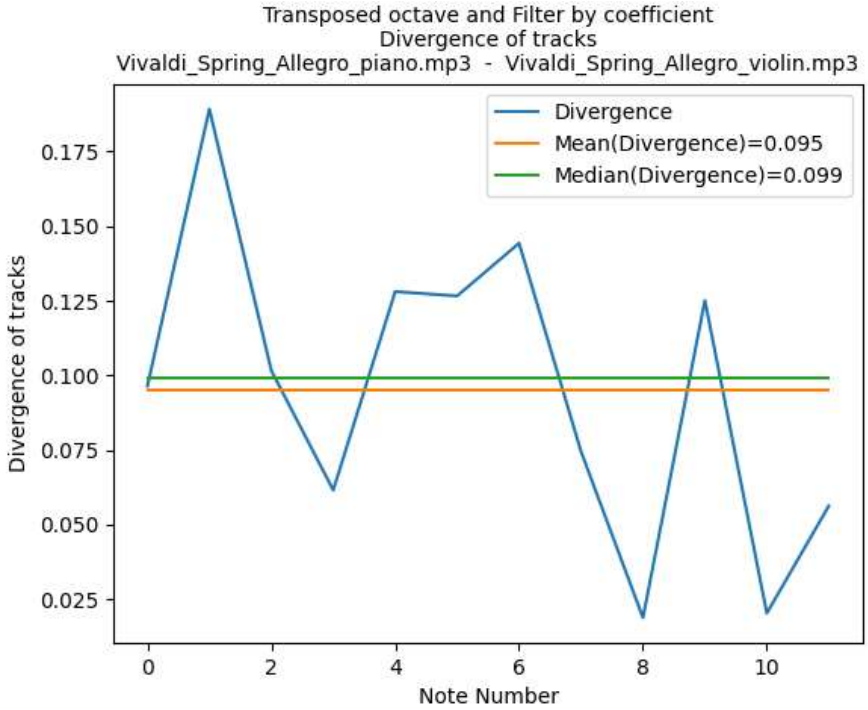

Рисунок 4.8 – Мера схожести между Vivaldi – Spring, Allegro (piano) и Vivaldi – Spring Allegro (violin)

Посчитаем меру схожести для каждой пары музыкальных произведений, таким образом, будет получено 36 результатов сравнений схожести композиций (8 из 36 сравнений приведены в таблице 4.1 и на рисунках 4.1-4.8).

Определим долю некорректных результатов, а именно – посчитаем для каждой пары похожих музыкальных композиций число пар не похожих произведений, у которых значение метрики оказалось меньше (то есть с точки зрения полученного результат такая пара оказалась более похожей), чем у рассматриваемой пары похожих треков. Просуммируем полученные результаты для каждой пары похожих композиций (для среднего и медианного значений в отдельности) и соотнесем с общим числом таких сравнений (рисунок 4.9).

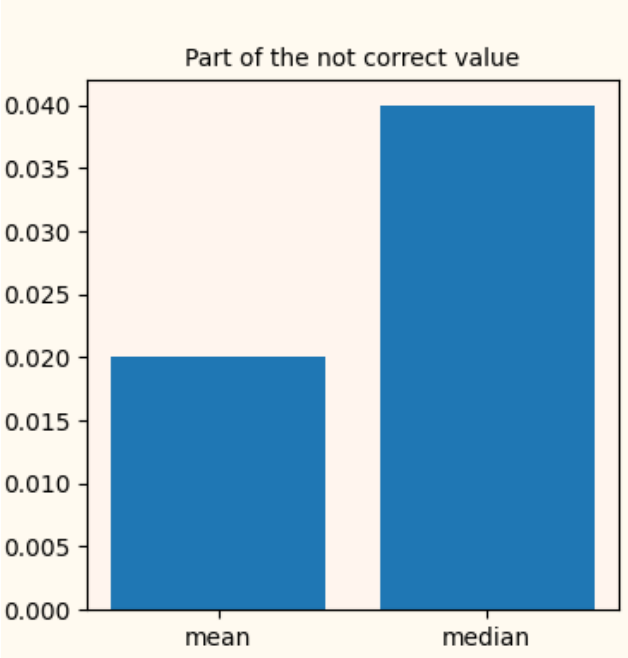

Рисунок 4.9 – Доля некорректных значений вычисленной метрики для тестового набора произведений

Далее определим долю некорректных результатов для различных значений коэффициента фильтрации, продемонстрировав тем самым влияние степени фильтрации на корректность получаемых значений схожести между музыкальными произведениями для метода транспонированной октавы (рисунок 4.10).

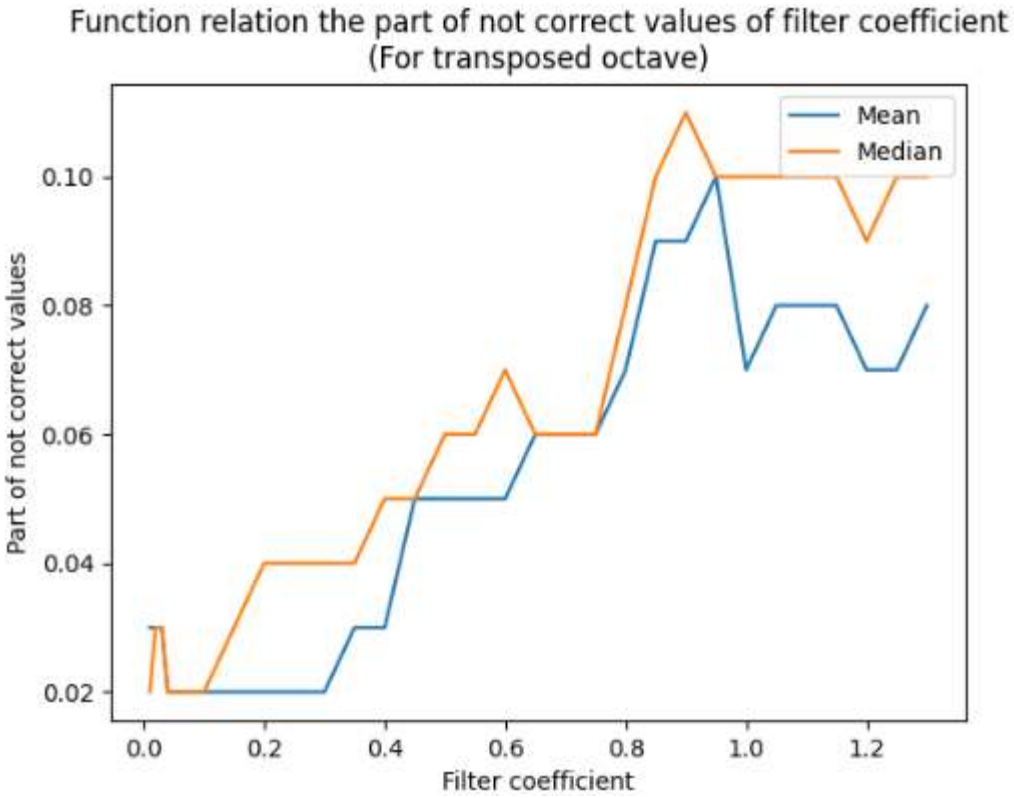

Рисунок 4.10 – Доля некорректных значений вычисленной метрики в зависимости от коэффициента фильтрации

По горизонтальной оси отложены значения коэффициента фильтрации, по вертикальной – доля некорректных значений метрики. Как видно из графиков, доля некорректных сравнений для медианного значения стабильно больше, нежели при использовании среднего значения, причем увеличение коэффициента заставляет расти долю некорректных результатов. Наибольшая степень корректности для среднего достигается при малых значениях коэффициента – до 0.25.

4.2 Фильтрация по числу фреймов и метод транспонированной октавы

Проведем тестирование работы алгоритма, который использует фильтрацию исходных кадров музыкального трека по заранее определенному фиксированному для всех треков числу фреймов. Также для формирования

нотного спектра данный алгоритм использует метод транспонированной октавы.

В качестве набора тестовых музыкальных произведений будут использоваться те же композиции, что и в пункте ранее. Попробуем оценить, насколько такой подход при фильтрации влияет на конечный результат, определив часть некорректных результатов вычисления метрики для всех пар из набора (36 возможных пар). Подход для определения числа некорректных результатов остается ровно таким же, какой был в пункте ранее при фильтрации по коэффициенту. Таким образом, посчитаем долю некорректных результатов для различных значений числа отбираемых фреймов при фильтрации, продемонстрировав тем самым влияние степени фильтрации на корректность получаемых значений схожести между музыкальными произведениями (рисунок 4.11).

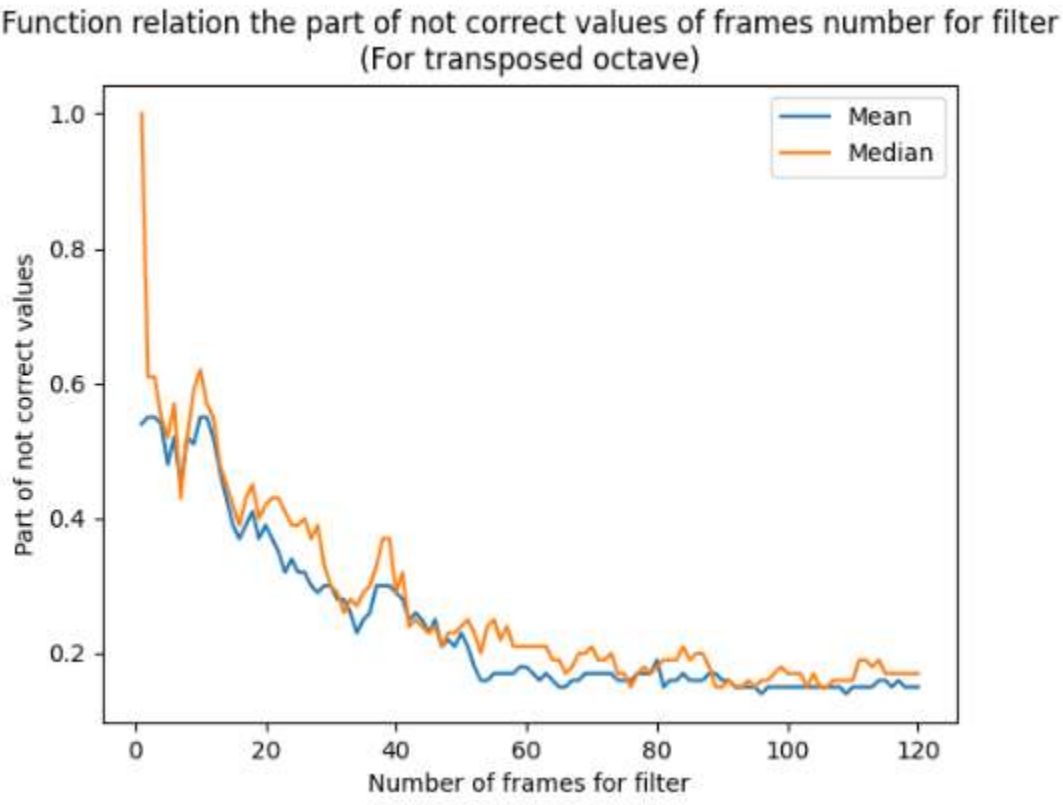

Рисунок 4.11 – Доля некорректных значений вычисленной метрики в зависимости от количества отбираемых фреймов для фильтрации

По горизонтальной оси отложено количество фреймов, отобранных после фильтрации, по вертикальной – доля некорректных значений метрики. Для каждого числа фреймов было произведено 36 сравнений музыкальных произведений (из описанного ранее набора композиций) для получения доли некорректных результатов, также рассматривались возможные принимаемые значения числа фреймов для фильтрации от 1 до 120. Таким образом, было произведено 4320 сравнений. Как видно из графиков, доля некорректных результатов при определении схожести для медианного значения, ровно как и для способа фильтрации по коэффициенту, стабильно больше, нежели при использовании среднего значения. Наибольшая степень корректности для среднего и медианного значений для числа фреймов от 60 и более.

4.3 Пороговая фильтрация и метод полного нотного спектра

Проведем тестирование работы алгоритма, который использует фильтрацию исходных кадров музыкального трека по коэффициенту (пороговая фильтрация), а также для формирования нотного спектра использует метод полного нотного спектра.

В качестве набора музыкальных композиций для тестирования данного алгоритма будет взят набор из 9 произведений, на котором проводилось тестирование для метода транспонированной октавы и обоих видов фильтрации. Аналогично предыдущим пунктам попытаемся оценить долю некорректных результатов вычисления метрики при использовании метода полного нотного спектра и фильтрации фреймов по коэффициенту. Подход для определения числа некорректных результатов остается ровно таким же, какой был в предыдущих пунктах. Таким образом, посчитаем долю некорректных результатов для различных значений коэффициента фильтрации при использовании метода полного нотного спектра, продемонстрировав тем самым

влияние степени фильтрации на корректность получаемых значений схожести между музыкальными произведениями (рисунок 4.12).

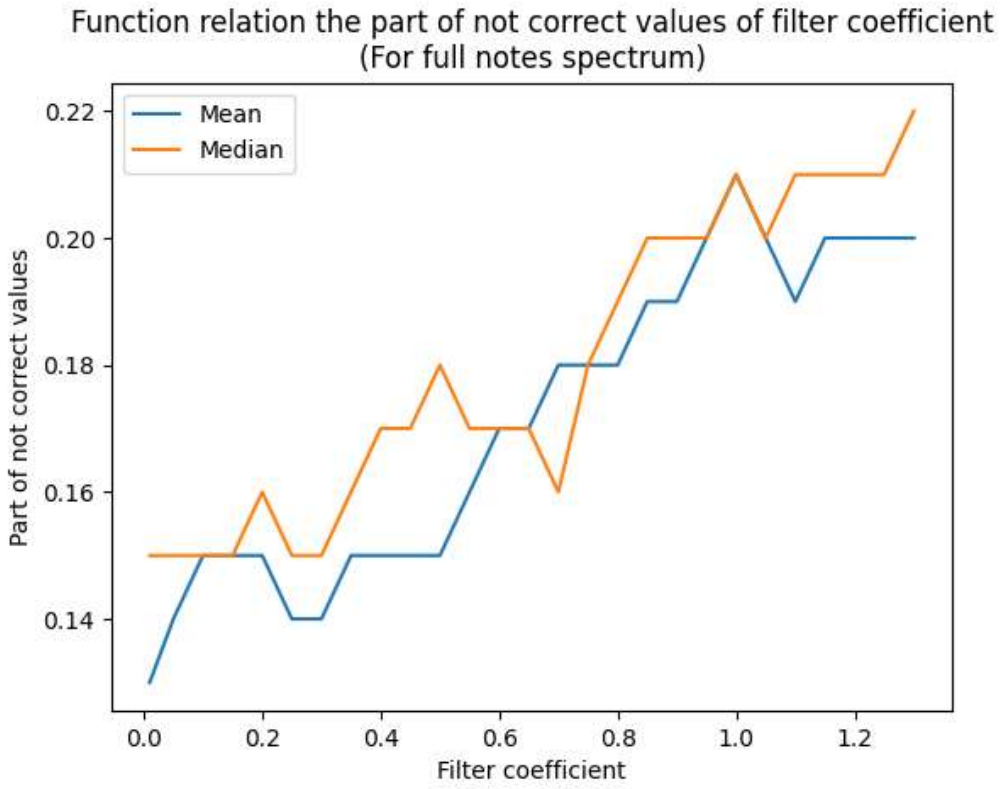

Рисунок 4.12 – Доля некорректных значений вычисленной метрики в зависимости от коэффициента фильтрации

По горизонтальной оси отложены значения коэффициента фильтрации, по вертикальной – доля некорректных значений метрики. Как видно из графика, доля некорректных вычислений метрики практически на всем рассматриваемом промежутке больше, нежели ошибка для среднего значения. Также стоит отметить, что, как и для метода транспонированной октавы, при увеличении значения коэффициента увеличивается и доля некорректных результатов, при этом величина доли некорректных результатов для метода полного спектра оказывается больше, чем для метода транспонированной октавы для всего рассматриваемого промежутка значений коэффициента. Наибольшая степень точности достигается при значениях коэффициента, близким к нулю. Кроме того, важно отметить тот факт, что метод полного нотного спектра производит работу со всеми 60 нотами в отдельности и, как показывают эм-

пирические данные, основная часть сигнала располагается приблизительно в первых 50 нотах для большинства музыкальных композиций (рисунок 4.13).

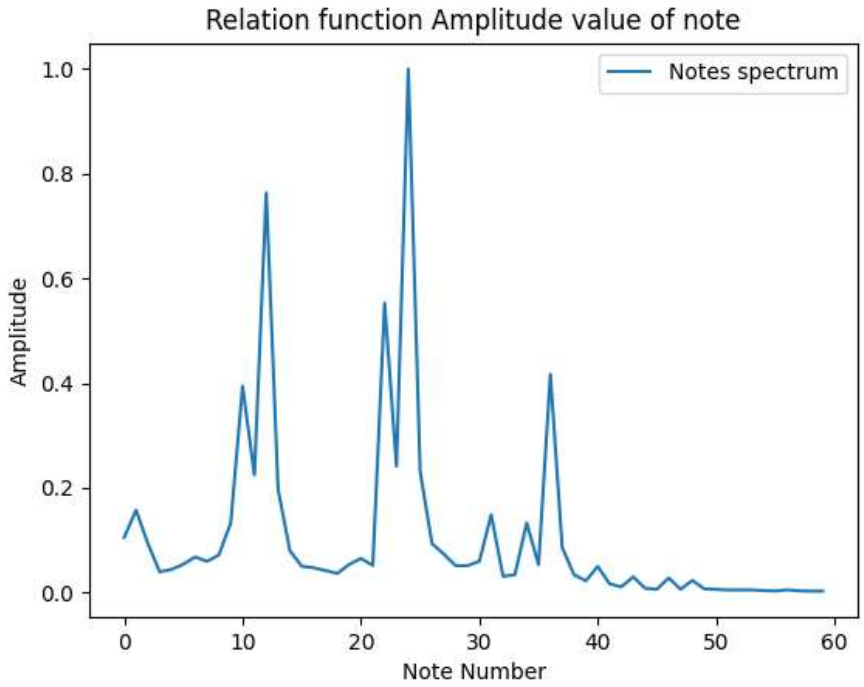

Рисунок 4.13 – Полный нотный спектр случайного фрейма случайного музыкального произведения

Данный график отражает типичный случай нотного спектра случайного фрейма случайного музыкального произведения. Как видно из приведенной зависимости, от 50 ноты среднее амплитудное значение оказывается равным 0 (или приблизительно равным нулю). При формировании матрицы плотности распределения вероятностей (МПРВ) данные ноты будут содержать большое количество нулевых значений (рисунок 4.14).

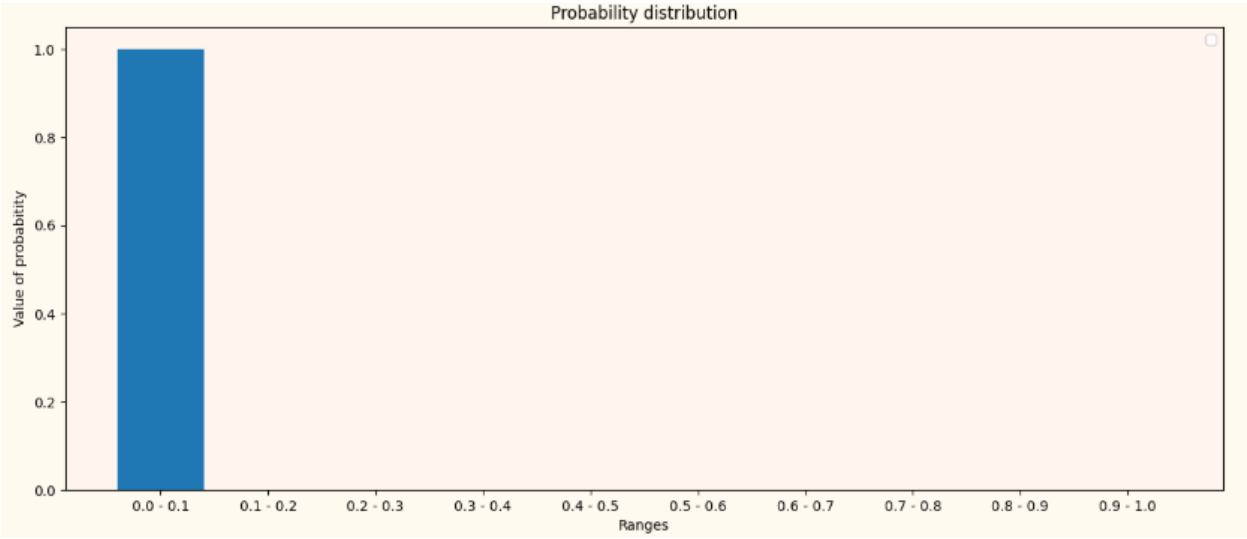

Рисунок 4.14 – Типичный случай распределения вероятностей для 50-ых нот случайного музыкального произведения

Данное распределение из 10 интервалов (значений) имеет единственную отличную от нуля вероятность. Таким образом, при подсчете расстояния Кульбака-Лейблера между такими нотами приходится вносить изменения в данные распределения и заменять нулевые значения на крайне малые, например, 0.0001, что, разумеется, влияет на результат вычисления расстояния Кульбака-Лейблера и, как следствие, на получаемое значение метрики схожести между музыкальными композициями.

Количество интервалов разбиения значений вероятностей (10) обусловлено тем, что при увеличении числа интервалов распределения будет происходить увеличение количества нулей в итоговом распределении вероятностей, что, как было описано ранее, влечет за собой изменения в получаемом результате.

4.4 Фильтрация по числу фреймов и метод полного нотного спектра

Проведем тестирование работы алгоритма, который использует фильтрацию исходных кадров музыкального трека по заранее определенному

фиксированному для всех треков числу фреймов, а также для формирования нотного спектра использует метод полного нотного спектра.

В качестве набора тестовых музыкальных произведений будут использоваться те же композиции, что и в пункте ранее. Попробуем оценить, насколько такой подход при фильтрации влияет на конечный результат, определив часть некорректных результатов вычисления метрики для всех пар из набора (36 возможных пар). Подход для определения числа некорректных результатов остается ровно таким же, какой был в пункте ранее при фильтрации по коэффициенту. Таким образом, посчитаем долю некорректных результатов для различных значений числа отбираемых фреймов при фильтрации, продемонстрировав тем самым влияние степени фильтрации на корректность получаемых значений схожести между музыкальными произведениями (рисунок 4.15).

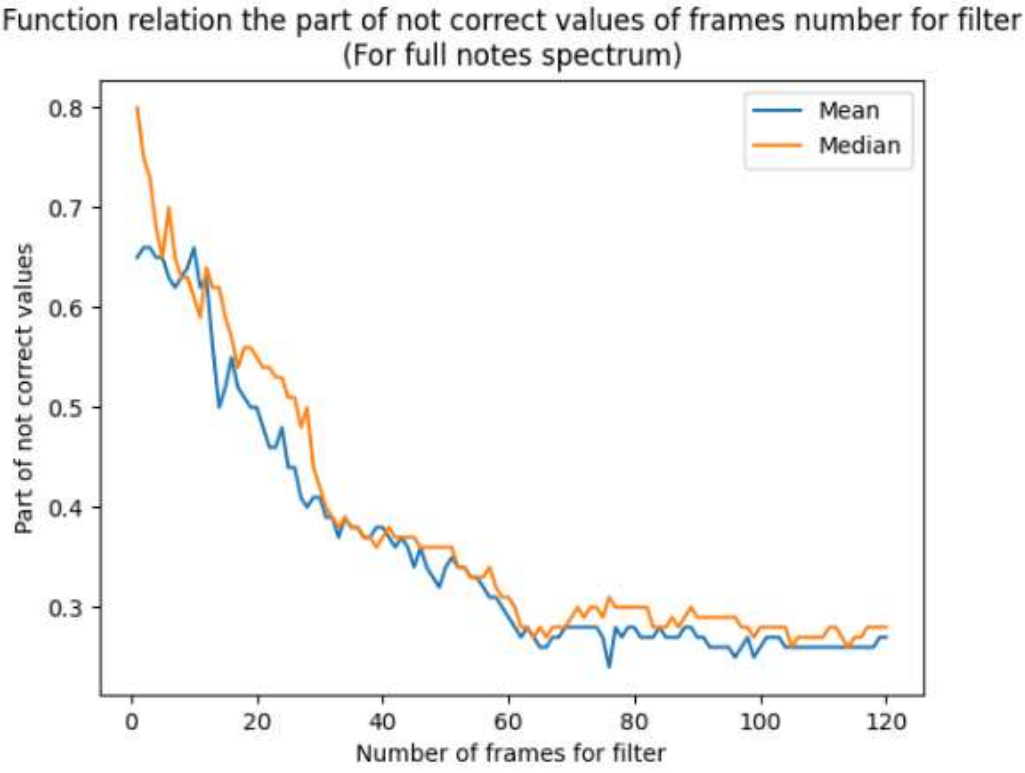

Рисунок 4.15 – Доля некорректных значений вычисленной метрики в зависимости от количества отбираемых фреймов для фильтрации

По горизонтальной оси отложено количество фреймов, отобранных после фильтрации, по вертикальной – доля некорректных значений метрики.

Для каждого числа фреймов было произведено 36 сравнений музыкальных произведений (из описанного ранее набора композиций) для получения доли некорректных результатов, также рассматривались возможные принимаемые значения числа фреймов для фильтрации от 1 до 120, таким образом, было произведено 4320 сравнений, ровно как и для фильтрация по числу фреймов при использовании метода транспонированной октавы. Как видно из графиков, доля некорректных результатов при определении схожести для медианного значения, ровно как и для способа фильтрации по коэффициенту, стабильно больше, нежели при использовании среднего значения, аналогично случаю при использовании метода транспонированной октавы, однако для данного метода доля некорректных результатов оказалось больше. Наибольшая степень корректности для среднего и медианного значений достигается для числа фреймов от 60 и более.

4.5 Выводы по экспериментальным данным

Полученные экспериментальные данные свидетельствуют о следующем:

1. Для всех рассмотренных видов фильтрации фреймов (по порогу и по количеству отбираемых фреймов) и всех видов получения нотного спектра (метода транспонированной октавы и метод полного нотного спектра) медианное значение между получающимися расстояниями Кульбака-Лейблера для плотностей распределения вероятностей музыкальных произведений дает большую долю некорректных значений определения метрики схожести композиций, нежели среднее значение. Таким образом, использование среднего значения является более предпочтительным на основании полученных экспериментальных результатов.

2. Фильтрация фреймов по порогу для рассмотренного диапазона значений дает меньшую долю некорректных вычислений метрики схожести между композициями, нежели фильтрация по числу фреймов. Причем данная оценка справедлива и для метода полного нотного спектра, и для метода транспонированной октавы. Кроме того, наилучшие значения для выбора коэффициента лежат от 0 до 0.2-0.3. Таким образом, фильтрация по коэффициенту является более предпочтительной на основании полученных результатов экспериментов и пункта 3.2.3.
3. Для всех рассмотренных видов фильтрации фреймов (по порогу и по количеству отбираемых фреймов) использование метода транспонированной октавы давало меньшую долю некорректных результатов вычисления метрики, чем при использовании метода полного нотного спектра, причем и для медианного значения и для среднего значения между получающимися расстояниями Кульбака-Лейблера для ПРВ музыкальных произведений. Таким образом, использование метода транспонированной октавы является более предпочтительным на основании полученных результатов экспериментов и пункта 3.3.3.

5 Разработка и стандартизация программных средств

В данном разделе будут описаны детали организации проектирования ПС по определению схожести музыкальных произведений, а именно – разработка и стандартизация реализуемого ПС [10].

5.1 Планирование работы проекта

Разработаем план проектных работ для ПС. Приведем план в виде диаграммы Ганта. Для условных обозначений в графе “Исполнители” приведем их расшифровки: А – дипломник, Б – научный руководитель, В – консультант. Диаграмма выполнения комплекса проектных работ с указанием сроков и исполнителей представлена в таблице 5.1.

Таблица 5.1 – Диаграмма Ганта выполнения комплекса проектных работ

№ работы	Название работы	Начало	Конец	Временные периоды (в единицах времени)													Исполнители
				I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII	
1	Формирование требований к ПС	01.03	08.03	—													А,Б
2	Выбор языка программирования	09.03	15.03		—												А
3	Изучение теоретической части прикладной области для реализации ПС	16.03	31.03			—	—										А,Б
4	Изучение библиотек выбранного языка программирования	22.03	31.03				—										А
5	Разработка концепции ПС	01.04	08.04					—									А,Б
6	Разработка предварительных проектных решений по ПС и его частям	01.04	15.04					—	—								А

Продолжение таблицы 5.1

7	Разработка программного кода для формирования нотного спектра музыкальных композиций	16.04	22.04							—							А
8	Разработка программного кода для вычисления плотности распределения вероятностей для нот музыкального произведения	23.04	30.04								—						А
9	Разработка программного кода для вычисления метрики Кульбака-Лейблера между музыкальными произведения	01.05	08.05									—					А
10	Испытания на соответствие техническому заданию написанного ПС	09.05	15.05										—				А

Продолжение таблицы 5.1

11	Экспериментальное определение оптимального алгоритма решения поставленной задачи	16.05	22.05											—			А,Б
12	Анализ результатов испытания ПС и устранение недостатков, выявленных при испытаниях	16.05	30.05												—		А
13	Стандартизация ПС	23.05	30.05													—	А,В

5.2 Определение затрат на выполнение и внедрение проекта ПС

Начнем расчет с определения месячной зарплаты проектировщиков. Основным разработчиком является дипломник. Коэффициент загрузки для него принимается равным единице ($К_{загр} = 1$). Средняя по стране зарплата в сфере ИТ на момент первой половины 2021 года равна 132 035 руб. в месяц. Выполним корректировку заработной платы в 3 раза в сторону уменьшения в связи с тем, что проектировщик (дипломник) является начинающим. Таким образом, приходим к зарплате равной 44 011 руб. в месяц.

Коэффициент загрузки для консультанта примем равным 0,03 ($К_{загр} = 0{,}03$). Согласно данным Росстата РФ средняя заработная плата преподавателей образовательных организаций высшего профессионального образования государственной и муниципальной форм собственности по субъектам Российской Федерации за январь-декабрь 2020 года по г. Санкт-Петербургу равна 120 526 руб. в месяц [11]. Аналогичную заработную плату возьмем для научного руководителя, однако коэффициент загрузки возьмем равным 0,07 ($К_{загр} = 0{,}07$).

Произведем вычисление полной дневной стоимости работы каждого специалиста согласно формуле:

$$С_{полн} = (Z_{зп}/Т_{ср}) \text{ x } (1 + Ф + Н) \text{ x } (1 + П) \text{ x } (1 + НДС) \text{ [руб./день]}$$

Примем среднемесячное число рабочих дней равным 20,69 ($Т_{ср}$=20,69).

Страховые взносы в государственные фонды, выплачиваемые работодателем как процент от суммы заработной платы равен 30,2% (Ф=30,2%).

Среднюю прибыль исполнителя П при выполнении проектов в сфере ИТ примем равной 15% (П=15%).

Процент накладных расходов Н, вычисляемый к фонду оплаты труда основных работников, для проектных организаций в сфере информатики возьмем равным 60% (Н=60%).

Ставка налога на добавленную стоимость НДС – 20%.

Таблица 5.2 – Полные затраты в день на дипломанта

Наименование статей	Ед. изм.	Нормативы/ Затраты	Примечание
Величина среднемесячной начисленной заработной платы специалиста ($Z_{зп}$)	руб./месяц	44011,00	Оклад сотрудника $Z_{зп}$
Среднемесячное количество рабочих дней ($T_{ср}$)	дней/месяц	20,69	Среднее за 2019-2022 годы
Расчет ставки специалиста в день:			
Тарифная дневная ставка ($Z_д$)	руб./день	2127,16	$Z_д = Z_{зп}/T_{ср}$
Страховые взносы 30,2% от суммы зарплаты работников ($C_{сд}$)	руб./день	642,40	$C_{сд}=Z_д$ х Ф
Дневная оплата работника с учетом страховых взносов ($Z_{дс}$)	руб./день	2759,56	$Z_{дс}= Z_д + C_{сд}$
Накладные расходы ($C_{нр}$)	руб./день	1276,30	$C_{нр}= Z_д$ х Н

Продолжение таблицы 5.2

Себестоимость одного человеко/дня ($С_{ч/д}$)	руб./день	4035,86	$С_{ч/д}= Z_{дс}+ С_{нр}$
Дневная прибыль ($С_{прд}$)	руб./день	605,38	$С_{прд}= С_{ч/д}$х П
Ставка специалиста без учета НДС ($С_{дсс}$)	руб./день	4641,24	$С_{дсс}= С_{прд}+ С_{ч/д}$
Величина налога на добавленную стоимость в расчете на день ($С_{ндс}$)	руб./день	928,25	$С_{ндс} = С_{дсс}$х НДС
Полная дневная стоимость работы специалиста ($С_{полн}$)	руб./день	5569,49	$С_{полн}= С_{дсс}+ С_{ндс}$

Далее вычислим аналогичную величину для консультанта и научного руководителя.

Однако стоит обратить внимание, что для данных расчетов процент накладных расходов Н, вычисляемого к фонду оплаты труда работников (консультанта и научного руководителя), берется равным 42% (Н=42%).

Таблица 5.3 – Полные затраты в день на консультанта

Наименование статей	Ед. изм.	Нормативы/ Затраты	Примечание
Величина среднемесячной начисленной заработной платы специалиста ($Z_{зп}$)	руб./месяц	120526,00	Оклад сотрудника $Z_{зп}$

Продолжение таблицы 5.3

Среднемесячное количество рабочих дней ($Т_{ср}$)	дней/месяц	20,69	Среднее за 2019-2022 годы
Расчет ставки специалиста в день:			
Тарифная дневная ставка ($Z_д$)	руб./день	5825,33	$Z_д = Z_{зп}/Т_{ср}$
Страховые взносы 30,2% от суммы зарплаты работников ($С_{сд}$)	руб./день	1759,25	$С_{сд}=Z_д$ х Ф
Дневная оплата работника с учетом страховых взносов ($Z_{дс}$)	руб./день	7584,58	$Z_{дс}= Z_д + С_{сд}$
Накладные расходы ($С_{нр}$)	руб./день	2446,64	$С_{нр}= Z_д$ х Н
Себестоимость одного человеко/дня ($С_{ч/д}$)	руб./день	10031,22	$С_{ч/д}= Z_{дс}+ С_{нр}$
Дневная прибыль ($С_{прд}$)	руб./день	1504,68	$С_{прд}= С_{ч/д}$ х П
Ставка специалиста без учета НДС ($С_{дсс}$)	руб./день	11535,90	$С_{дсс}= С_{прд}+ С_{ч/д}$
Величина налога на добавленную стоимость в расчете на день ($С_{ндс}$)	руб./день	2307,18	$С_{ндс} = С_{дсс}$ х НДС

Продолжение таблицы 5.3

Полная дневная стоимость работы специалиста ($С_{полн}$)	руб./день	13843,08	$С_{полн} = С_{дсс} + С_{ндс}$

Выполним расчет для научного руководителя.

Таблица 5.4 – Полные затраты в день на научного руководителя

Наименование статей	Ед. изм.	Нормативы/ Затраты	Примечание
Величина среднемесячной начисленной заработной платы специалиста ($Z_{зп}$)	руб./месяц	120526,00	Оклад сотрудника $Z_{зп}$
Среднемесячное количество рабочих дней ($Т_{ср}$)	дней/месяц	20,69	Среднее за 2019-2022 годы
Расчет ставки специалиста в день:			
Тарифная дневная ставка ($Z_д$)	руб./день	5825,33	$Z_д = Z_{зп}/Т_{ср}$
Страховые взносы 30,2% от суммы зарплаты работников ($С_{сд}$)	руб./день	1759,25	$С_{сд}=Z_д$ х Ф
Дневная оплата работника с учетом страховых взносов ($Z_{дс}$)	руб./день	7584,58	$Z_{дс}= Z_д + С_{сд}$

Продолжение таблицы 5.4

Накладные расходы ($С_{нр}$)	руб./день	2446,64	$С_{нр}= Z_{д}$х Н
Себестоимость одного человеко/дня ($С_{ч/д}$)	руб./день	10031,22	$С_{ч/д}= Z_{дс}+ С_{нр}$
Дневная прибыль ($С_{прд}$)	руб./день	1504,68	$С_{прд}= С_{ч/д}$х П
Ставка специалиста без учета НДС ($С_{дсс}$)	руб./день	11535,90	$С_{дсс}= С_{прд}+ С_{ч/д}$
Величина налога на добавленную стоимость в расчете на день ($С_{ндс}$)	руб./день	2307,18	$С_{ндс} = С_{дсс}$х НДС
Полная дневная стоимость работы специалиста ($С_{полн}$)	руб./день	13843,08	$С_{полн}= С_{дсс}+ С_{ндс}$

Далее перейдем к расчету цены проекта$С_{пр}$по формуле:

$$С_{пр}=\sum_{i}^{n} С_{полн\,i} * Т_{i} * К_{загр\,i},$$

где:

$С_{полн\,i}$ – полная дневная стоимость работы i-ого специалиста [руб./день];

$Т_{i}$ – время участия i-ого специалиста в работе над проектом [дней];

$К_{загр\,i}$ – коэффициент загрузки i-ого специалиста работами в проекте;

n – число специалистов, занятых в проекте.

Число специалистов занятых в проекте (n) равно 3 – дипломник, научный руководитель, консультант.

Полные дневные стоимости работ всех специалистов были вычислены ранее (см. табл. 2.1, табл. 2.2, табл. 2.3).

Коэффициенты загрузки ($К_{загр}$) для специалистов были приняты следующими: дипломник – 1, научный руководитель – 0,07, консультант – 0,03.

Примем стандартный календарь, при котором 1 ставка – 8 часов в день с 5-дневной рабочей неделей. В таком случае, согласно календарю работ, время участия консультанта в проекте равняется 6 дням. Время участия научного руководителя равно 30 дней, а дипломника – 66 дней.

Таким образом, цена проекта будет равняться $С_{пр}$= 399148,75 руб.

5.3 Определение кода разрабатываемого программного изделия

Определим для разработанного в рамках программного средства код в соответствии с действующим классификатором ОКПД 2 [12].

Разрабатываемое ПС классифицируется следующим образом:

- ПС относится к программным продуктам и услугам по разработке программного обеспечения; консультационные и аналогичные услуги в области информационных технологий ⇒ код 62;
- ПС относится к программным продуктам и услугам по разработке программного обеспечения; консультационные и аналогичные услуги в области информационных технологий ⇒ код 62.0;
- ПС относится к продуктам программным и услугам по разработке и тестированию программного обеспечения ⇒ код 62.01;
- ПС относится к услугам по проектированию, разработке информационных технологий для прикладных задач и тестированию программного обеспечения ⇒ код 62.01.1;
- ПС относится к услугам по проектированию, разработке информационных технологий для прикладных задач и тестированию программного обеспечения ⇒ код 62.01.11;

Таким образом, описав каждый шаг классификации ПС, разработанный продукт относится к классу 62.01.11согласно классификатору ОКПД 2.

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе был произведен обзор предметной области, а именно – анализ существующих решений по данной теме и их сравнение с разрабатываемым вариантом. Кроме того, были составлены и описаны 4 модификации алгоритма, с помощью которых определялась метрика схожести музыкальных произведений:

- С помощью метода транспонированной октавы и фильтрацией по числу фреймов;
- С помощью метода транспонированной октавы и фильтрацией по коэффициенту;
- С помощью метода полного нотного спектра и фильтрацией по числу фреймов;
- С помощью метода полного нотного спектра и фильтрацией по коэффициенту;

Принцип работы модификаций алгоритма был подробно описан. Также для каждого из способов было произведено тестирование, на основание которого был сделан вывод о том, какой из представленных вариантов алгоритма наилучшим образом производит определение метрики. Все 4 способа определения схожести между музыкальными произведениями были программно реализованы.

Развитием данной работы может служить модификация разработанного алгоритма на основе экспериментальных данных так, чтобы метрика схожести музыкальных произведений давала наиболее корректные результаты для любых жанров и типов музыки. Кроме того, возможным представляется обучение нейронной сети не только для определения метрики схожести, но и получения ответа о том, являются ли рассматриваемые композиции похожими (одинаковыми) на основе полученных ранее знаний.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Поздняков С. Н. Математические методы цифровой обработки сигналов: учеб. пособие / С. Н. Поздняков, С. В. Рыбин. – СПб.: изд-во СПбГЭТУ «ЛЭТИ», 2015. – 61 с.
2. Weller-Fahy D. J., Borghetti B. J., Sodemann A. A. A Survey of Distance and Similarity Measures Used Within Network Intrusion Anomaly Detection // IEEE Communications Surveys and Tutorials. 2015. Vol. 17, No. 1.
3. Wang A. L. An Industrial-Strength Audio Search Algorithm [Электронный ресурс]. – Режим доступа: https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf. – (Дата обращения: 10.04.2021).
4. Архитектура и алгоритмы индексации аудиозаписей [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/vk/blog/330988/. – (Дата обращения 10.04.2021).
5. Фихтенгольц Г. М. Курс дифференциального и интегрального исчисления Т.3: учеб. для вузов / Г. М. Фихтенгольц. – СПб.: изд-во Лань, 2009. – 656 с.
6. Дауни А. Б. Цифровая обработка сигналов на языке Python / Э. А. Бряндинский. – М.: изд-во ДМК-Пресс, 2017. – 162 с.
7. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов / И. И. Грушко. – М.: изд-во Мир, 1989. – 448 с.
8. Рашка С. Python и машинное обучение / С. Рашка. – М.: изд-во ДМК-Пресс, 2017. – 418 с.
9. Николенко С. И. Глубокое обучение / С. И. Николенко, А. А. Кадурин, Е. О. Архангельская. – М.: изд-во Питер, 2018. – 480 с.

10. Фомин В. И. Методические указания по выполнению дополнительного раздела Разработка и стандартизации программных средств при выполнении выпускной квалификационной работы бакалавра: метод. указания / В. И. Фомин. – СПб.: изд-во СПбГЭТУ «ЛЭТИ», 2020. – 39с.
11. Рынок труда, занятость и заработная плата [Электронный ресурс]. – Режим доступа: https://rosstat.gov.ru/labor_market_employment_salaries#. – (Дата обращения 20.04.2021).
12. Общероссийские классификаторы [Электронный ресурс]. – Режим доступа: https://classifikators.ru/okpd. – (Дата обращения 21.04.2021).

ПРИЛОЖЕНИЕ А

Исходный код программы

```
# Класс обработки музыкального трека
class ProcessMusicTrack():
    F0 = 220                                                    # Частота первой ноты
    FRAME_SIZE = 8192                                           # Кол-во точек в фрейме
    THRESHOLD_OF_NOISE = 0.25                                   # Коэффициент фильтрации
    NUMBER_OF_NOTES = 60                                        # Кол-во рассматриваемых нот
    NUMBER_OF_GOOD_FRAMES = 60                                  # Кол-во рассматриваемых фреймов
    SAMPLE_RATE = 44100                                         # Частота дискретизации
    NUMBER_OF_NOTES_AT_OCTAVE = 12                              # Кол-во нот в октаве
    NUMBER_OF_OCTAVES = 5                                       # Кол-во октав
    STEP_FOR_DISTRIBUTION_DENSITY_OF_NOTES = 0.1                # Шаг вероят-
                                                                # ностного распределения
    LOWER_DENSITY_LIMIT = 0.0001                                # Порог снизу для вероятностей
    SPECTRUM_MODE = None                                        # Режим формирования нотного спектра
    FILTER_MODE = None                                          # Режим фильтрации


    # Создает объект обработки трека по умолчанию
    # self - объект класса
    # path_to_track - путь к файлу
    # spectrum_mode - режим формирования нотного спектра
    # filter_mode - Режим фильтрации
    def __init__(self, path_to_track, spectrum_mode, filter_mode):
        self.music_track = MusicTrack(path_to_track)
        self.output_directory = path_to_track[0:len(path_to_track)-4]
        self.numberOfFrames = -1
        self.frames = []
        self.fourier_frames = []
        self.frequencies = []
        self.notes_indexes = []
        self.notes_environments = []
        self.spectral_matrix = []
        self.distribution_density_for_notes = []
        ProcessMusicTrack.SPECTRUM_MODE = spectrum_mode.value
        ProcessMusicTrack.FILTER_MODE = filter_mode.value

    # Фильтрация фреймов по коэффициенту
    # self - объект класса
    # return - отфильтрованные фреймы
    def filter_frames_by_coefficient(self):
        frames = []
        for i in range(self.numberOfFrames):
            frame =
            np.abs(self.music_track.music_data[ProcessMusicTrack.FRAME_SIZE *
            i:ProcessMusicTrack.FRAME_SIZE * (i + 1)])
            if np.mean(frame) >= ProcessMusicTrack.THRESHOLD_OF_NOISE *
                self.music_track.middleValue:
                frame = self.music_track.music_data[ProcessMusicTrack.FRAME_SIZE
                        * i:ProcessMusicTrack.FRAME_SIZE * (i + 1)]
                frames += [frame]
            return frames
```

```
# Фильтрация фреймов по числу
# self - объект класса
# return - отфильтрованные фреймы
def filter_frames_by_number(self):
    frames = []
    indexes_and_means_of_good_frames = {a * -1: a * -1 for a in range(1,
                            ProcessMusicTrack.NUMBER_OF_GOOD_FRAMES + 1)}
    for i in range(self.numberOfFrames):
        frame =
        np.abs(self.music_track.music_data[ProcessMusicTrack.FRAME_SIZE *
        i:ProcessMusicTrack.FRAME_SIZE * (i + 1)])
        self.check_frame(indexes_and_means_of_good_frames, frame, i)
        for index, value in indexes_and_means_of_good_frames.items():
        frame = self.music_track.music_data[ProcessMusicTrack.FRAME_SIZE *
                index:ProcessMusicTrack.FRAME_SIZE * (index + 1)]
        frames += [frame]
    return frames

# Выполнение преобразования Фурье
# self - объект класса
# return - результат преобразования Фурье, набор частот
def fourier_transformation(self):
    fourier_frames = []
    xf = rfftfreq(ProcessMusicTrack.FRAME_SIZE, 1 /
                  ProcessMusicTrack.SAMPLE_RATE)
    for i in range(len(self.frames)):
        yf = rfft(self.frames[i])
        fourier_frames = fourier_frames + [np.abs(yf)]
    return fourier_frames, xf

# Формирование нот по диапазонам частот
# self - объект класса
# return - индексы нот, индексы-границы для каждой ноты
def make_notes(self):
    notes_indexes = []
    notes_environments = []
    for i in range(ProcessMusicTrack.NUMBER_OF_NOTES):
        delta = 100000
        freq = -1
        index = -1
        fi = ProcessMusicTrack.F0 * (2 ** (i / 12))
        for current_index in range(len(self.frequencies)):
            if abs(self.frequencies[current_index] - fi) <= delta:
                delta = abs(self.frequencies[current_index] - fi)
                freq = self.frequencies[current_index]
                index = current_index
            else:
                break
        notes_indexes.append(index)
        if i > 0:
            middle = (self.frequencies[index] +
                      self.frequencies[notes_indexes[len(notes_indexes) - 2]]) / 2
            k = notes_indexes[len(notes_indexes) - 2]
            while middle >= self.frequencies[k]:
                k += 1
            notes_environments[i - 1].append(k - 1)
            notes_environments.append([k])
            if i == ProcessMusicTrack.NUMBER_OF_NOTES - 1:
                k = index
                while self.frequencies[index] + self.frequencies[index] - middle >=
                                                                    self.frequencies[k]:
                    k += 1
```

```
            notes_environments[i].append(k - 1)
            middle = (self.frequencies[notes_indexes[0]] +
                      self.frequencies[notes_indexes[1]]) / 2
            k = notes_indexes[0]
            while (self.frequencies[notes_indexes[0]] - (middle –
                   self.frequencies[notes_indexes[0]])) < self.frequencies[k]:
               k -= 1
            notes_environments[0].append(notes_environments[0][0])
            notes_environments[0][0] = k + 1
        else:
            notes_environments.append([])
    return notes_indexes, notes_environments

# Формирование матрицы полного спектра
# self - объект класса
# return - матрица полного нотного спектра
def make_matrix_of_full_spectrum(self):
    matrix = []
    for frame in self.fourier_frames:
        matrix.append([])
        for note_environment in self.notes_environments:
            note_mid = np.mean(frame[note_environment[0]:
                                     note_environment[1] + 1])
            matrix[len(matrix) - 1].append(note_mid)
        matrix[len(matrix) - 1] /= max(matrix[len(matrix) - 1])

    return matrix

# Формирование матрицы спектра по методу транспонированной октавы
# self - объект класса
# return - матрица нотного спектра
def make_matrix_of_transposed_octave(self):
    matrix = []
    for frame in self.fourier_frames:
        matrix.append([])
        for note_number in
            range(ProcessMusicTrack.NUMBER_OF_NOTES_AT_OCTAVE):
            notes_mid = []
            for octave_number in
                range(ProcessMusicTrack.NUMBER_OF_OCTAVES):
                notes_mid.append(np.mean(frame[self.notes_environments[
                                 octave_number *
                                 self.NUMBER_OF_NOTES_AT_OCTAVE +
                                 note_number][0]:
                                   self.notes_environments[
                                      octave_number *
                                      self.NUMBER_OF_NOTES_AT_OCTAVE +
                                      note_number][1] + 1]))
            matrix[len(matrix) - 1].append(np.mean(notes_mid))
        matrix[len(matrix) - 1] /= max(matrix[len(matrix) - 1])
    return matrix

# Формирование плотностей распределения вероятностей
# self - объект класса
# return - матрица распределения вероятностей
def get_distribution_density_for_notes(self):
    distribution_density_for_notes = []
    for j in range(len(self.spectral_matrix[0])):
        distribution_density_for_note = [0 for c in range(int(1 /
        ProcessMsicTrack.STEP_FOR_DISTRIBUTION_DENSITY_OF_NOTES))]
        for i in range(len(self.spectral_matrix)):
```

```
            if self.spectral_matrix[i][j] != 1:
                distribution_density_for_note[int(self.spectral_matrix[i][j] //
                ProcesicTrack.STEP_FOR_DISTRIBUTION_DENSITY_OF_NOTES)]
                 += 1
            else:
                distribution_density_for_note[len(distribution_density_for_note) - 1]
                 += 1
        distribution_density_for_note = [el / len(self.spectral_matrix)
                                                for el in distribution_density_for_note]
            distribution_density_for_notes.append(distribution_density_for_note)
    return distribution_density_for_notes

# Обработка музыкального произведения
# self - объект класса
# return - None
 def start_processing(self):
        if not os.path.exists(self.output_directory):
            os.mkdir(self.output_directory)
        self.music_track.prepare_data()
        self.music_track.load_music_data()
        self.numberOfFrames = self.music_track.supplement_signal()
        if ProcessMusicTrack.FILTER_MODE == FilterMode.BY_NUMBER.value:
            self.frames = self.filter_frames_by_number(False)
        else:
            self.frames = self.filter_frames_by_coefficient(False)
        print('Number of good frames: ' + str(len(self.frames)))
        print('Length of frames: ' + str(len(self.frames[0])))
        print('-------------------------------------------------')

        self.fourier_frames, self.frequencies = self.fourier_transformation(False)
        self.notes_indexes, self.notes_environments = self.make_notes(False)
        #self.print_notes()
        print('-------------------------------------------------------------')

        if ProcessMusicTrack.SPECTRUM_MODE ==
                SpectrumMode.FULL_SPECTRUM.value:
            self.spectral_matrix = self.make_matrix_of_full_spectrum()
        else:
            self.spectral_matrix = self.make_matrix_of_transposed_octave()
        print('Number of matrixes rows: ' + str(len(self.spectral_matrix)))
        print('Number of matrixes columns: ' + str(len(self.spectral_matrix[0])))
        self.output_to_file_spectral_matrix()
        self.output_graphic_of_spectral_matrix(2)

        self.distribution_density_for_notes = self.get_distribution_density_for_notes()
        self.output_to_file_distribution_density()

    # Определение расстояние Кульбака-Лейблера
    # self - объект класса
    # second_track - второе музыкальное произведение
    # return - расстояние Кульбака-Лейблера
    def get_Kullback_Leibler_divergence(self, second_track):
        kullback_leibler_divergencies = []
        for i in range(len(self.distribution_density_for_notes)):
            divergence=0
            for j in range(len(self.distribution_density_for_notes[i])):
                q = second_track.distribution_density_for_notes[i][j]
                if second_track.distribution_density_for_notes[i][j] == 0:
                    q = ProcessMusicTrack.LOWER_DENSITY_LIMIT
                if self.distribution_density_for_notes[i][j] != 0:
                    divergence += self.distribution_density_for_notes[i][j] *
                                  math.log2(self.distribution_density_for_notes[i][j] / q)
```

```
            kullback_leibler_divergencies.append(divergence)
        return kullback_leibler_divergencies

# Определение среднего значения дивергенций
# self - объект класса
# first_divergence - первая дивергенция
# second_divergence - первая дивергенция
# return - среднее значение дивергенций
def get_mean_of_divergencies(self, first_divergence, second_divergence):
        mean_of_divergencies = []
        for i in range(len(first_divergence)):
            mean_of_divergencies.append(np.mean([first_divergence[i],
                                                 second_divergence[i]]))

        return mean_of_divergencies



# Определение схожести музыкальных произведений
# self - объект класса
# track1 - первый трек
# track2 - второй трек
# return - значение схожести треков
 def get_divergence_of_tracks(self, track1, track2):
        first_divergence = track1.get_Kullback_Leibler_divergence(track2)
        second_divergence = track2.get_Kullback_Leibler_divergence(track1)
        ProcessMusicTrack.show_divergence(track1,
                                          track1.music_track.short_track_name,
                                           track2.music_track.short_track_name,
                                           first_divergence, 'K-L Divergence')
        ProcessMusicTrack.show_divergence(track2,
                                                   track2.music_track.short_track_name,
                                                    track1.music_track.short_track_name,
                                                    second_divergence, 'K-L Divergence')
                                                     mean_of_divergencies
        ProcessMusicTrack.get_mean_of_divergencies(None,first_divergence,
                                                               second_divergence)
        ProcessMusicTrack.show_divergence(track1,
                                                  track1.music_track.short_track_name,
                                                  track2.music_track.short_track_name,
                                                  mean_of_divergencies,
                                                  'Divergence of tracks')

        return [np.mean(mean_of_divergencies),
                statistics.median(mean_of_divergencies)]

# Перечисление для метода формирования спектра
class SpectrumMode(Enum):
    FULL_SPECTRUM = 'Full spectrum'
    TRANSPOSED_OCTAVE = 'Transposed octave'

# Перечисление для метода фильтрации
class FilterMode(Enum):
    BY_COEFFICIENT = 'Filter by coefficient'
    BY_NUMBER = 'Filter by number'
```

```
# Класс музыкальной композиции
class MusicTrack():
    # Создает объект музыкального трека по умолчанию
    # self - объект класса
    # path_to_track - путь к файлу
    def __init__(self, path_to_track):
        self.track_name = path_to_track
        self.short_track_name = re.search(r'[^\\]*\.[^\\]{1,}$', str(path_to_track)).group(0)

    # Конвертирует музыкальный трек в формат wav из mp3
    # self - объект класса
    # track_name_mp3 - имя mp3 трека
    # return - путь к wav файлу трека
    def mp3_to_wav(self, track_name_mp3):
        result = re.search(r'[^\\]*\.mp3$', track_name_mp3)
        track_name_wav = result.group(0)[0:len(result.group(0)) - 3] + 'wav'

        sound = AudioSegment.from_mp3(track_name_mp3)
        path_to_export = track_name_mp3[0:len(track_name_mp3)-4] + '\\' + track_name_wav
        sound.export(path_to_export, format='wav')

        return path_to_export

    # Конвертирует музыкальный трек в формат wav из ogg
    # self - объект класса
    # track_name_ogg - имя ogg трека
    # return - путь к wav файлу трека
    def ogg_to_wav(self, track_name_ogg):
        result = re.search(r'[^\\]*\.ogg$', track_name_ogg)
        track_name_wav = result.group(0)[0:len(result.group(0)) - 3] + 'wav'
        sound_data, _ = soundfile.read(track_name_ogg)
        path_to_export = track_name_ogg[0:len(track_name_ogg) - 4] + '\\' + track_name_wav
        soundfile.write(path_to_export, sound_data, ProcessMusicTrack.SAMPLE_RATE)
        return path_to_export

    # Определение формата входного трека и его преобразование
    # self - объект класса
    # return - пустое значение
    def prepare_data(self):
        result = re.search(r'\.mp3$', self.track_name)
        if result:
            print('Type of file: MP3')
            self.track_name = self.mp3_to_wav(self.track_name)
            return
        else:
            result = re.search(r'\.wav$', self.track_name)
            if result:
                print('Type of file: WAV')
                self.track_name = self.track_name
                return
            else:
                result = re.search(r'\.ogg$', self.track_name)
                if result:
                    print('Type of file: OGG')
                    self.track_name = self.ogg_to_wav(self.track_name)
                    return
                else:
                    print('ERROR: Unknown type of file')

        sys.exit()
```

```
# Рисование графика сигнала во временной области
# self - объект класса
# music_data - музыкальное произведение
# music_data - частота дискретизации
# return - None
def monophonic_graphic(self, music_data, sr):
    plt.figure(figsize=(7, 5))
    librosa.display.waveplot(music_data, sr=sr)
    plt.title('Monophonic')
    plt.show()

# Загрузка данных музыкального произвдения
# self - объект класса
# return - None
def load_music_data(self):
    self.music_data, sr = librosa.load(self.track_name, sr=ProcessMusicTrack.SAMPLE_RATE)
    self.music_data = librosa.to_mono(self.music_data)
    print('Length of track at points: ' + str(len(self.music_data)))
    print('Sampling frequency: ' + str(sr))
    #self.monophonic_graphic(self.music_data, sr)
    self.middleValue = np.mean(np.abs(self.music_data))
    print('Middle value module of signal: ' + str(self.middleValue))

# Доопределение сигнала нулями
# self - объект класса
# return - Сиглнал, дооперделенный нулями
def supplement_signal(self):
    self.music_data = list(self.music_data) + list([c * 0 for c in range(ProcessMusicTrack.FRAME_SIZE –
                                                    (len(self.music_data) -  ProcessMusicTrack.FRAME_SIZE *
                                                    (len(self.music_data) // ProcessMusicTrack.FRAME_SIZE)))])
    print('Length of track at points after supplement: ' + str(len(self.music_data)))
    numberOfFrames = int(len(self.music_data) / ProcessMusicTrack.FRAME_SIZE)
    print('Number of frames: ' + str(numberOfFrames) + '\n')
    return numberOfFrames
```