|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

**Факультет Информатика и системы управления (ИУ)**

**Кафедра Информационные системы и телекоммуникации (ИУ3)**

**Отчет**

**по лабораторной работе № 3**

**по курсу «Мультимедиа технологии»**

**на тему «Разработка мультимедийного приложения для сжатия и воспроизведения звука»**

Выполнил: Бободжанов А.Н.

Группа: ИУ3-31М

Проверил: Видьманов Д. А.

Москва, 2022 г.

**Содержание**

[**1 Задание** 3](#_Toc118113895)

[**2 Теоритическая часть** 4](#_Toc118113896)

[**3 Практическая часть** 7](#_Toc118113897)

[**Вывод** 7](#_Toc118113898)

[**Листинг** 7](#_Toc118113899)

**1 Задание**

1. Установить программную среду Anaconda Navigator или любой предпочитаемый IDE для работы с Python;
2. Запустить IDE. В случае использования Anaconda Navigator: запустить Anaconda Navigator. Запустить Spider;
3. Подготовить Wav файл для сжатия;
4. Напечатать и выполнить код, приведенный в методическом указании.

**Цель:** Иузчить структуру Wav файла, методы сжатия звука, метод главных компонент и реализовать сжатие Wav файла на языке Python.

**2 Теоритическая часть**

## Сжатие без потерь

В исходных данных находят какую-либо закономерность и с учётом этой закономерности генерируют вторую последовательность, которая однозначно описывает исходную. Например, для кодирования двоичных последовательностей, в которых много нулей и мало единиц, мы можем использовать такую замену:

00> 0  
01> 10  
10> 110  
11> 111

В таком случае шестнадцать битов:

00 01 00 00 11 10 00 00

будут преобразованы в тринадцать битов:

0 10 0 0 111 110 0 0

Если мы запишем сжатую строку без пробелов, мы всё равно сможем расставить в ней пробелы — а значит, восстановить исходную последовательность.

FLAC (Free Lossless Audio Codec — свободный аудио-кодек без потерь)

Принцип кодирования: алгоритм пытается описать сигнал такой функцией, чтобы полученный после её вычитания из оригинала результат (называемый разностью, остатком, ошибкой) можно было закодировать минимальным количеством битов.

Когда модель подобрана, алгоритм вычитает приближение из оригинала, чтобы получить остаточный (ошибочный) сигнал, который затем кодируется без потерь.

## Сжатие с потерями (MP3, AAC, WMA, OGG)

Используется алгоритм сжатия с потерями, размер MP3-файла со средним битрейтом 128 кбит/с примерно равен 1/11 от оригинального файла с аудио CD (несжатое аудио формата CD-Audio имеет битрейт 1411,2 кбит/с). MP3 файлы могут создаваться с высоким или низким битрейтом, что влияет на качество результата.

Принцип сжатия заключается в снижении точности некоторых частей звукового потока, что практически неразличимо для слуха большинства людей. Звуковой сигнал разбивается на равные по продолжительности отрезки, каждый из которых после обработки упаковывается в свой фрейм (кадр). Разложение в спектр требует непрерывности входного сигнала, в связи с этим для расчётов используется также предыдущий и следующий фрейм. В звуковом сигнале есть гармоники с меньшей амплитудой и гармоники, лежащие вблизи более интенсивных — такие гармоники отсекаются, так как среднестатистическое человеческое ухо не всегда сможет определить присутствие либо отсутствие таких гармоник. Такая особенность слуха называется эффектом маскировки. Также возможна замена двух и более близлежащих пиков одним усреднённым (что, как правило, и приводит к искажению звука). Критерий отсечения определяется требованием к выходному потоку. Поскольку весь спектр актуален, высокочастотные гармоники не отсекаются, а только выборочно удаляются, чтобы уменьшить поток информации за счёт разрежения спектра. После спектральной «зачистки» применяются математические методы сжатия и упаковка во фреймы.

### Эффект маскировки

В определённых случаях один звук может быть скрыт другим звуком. Например, разговор рядом с железнодорожными путями может быть совершенно невозможен, если мимо проезжает поезд. Такого рода эффекты называют маскировкой. Говорят, что слабый звук маскируется, если он становится неразличимым в присутствии более громкого звука.

### Одновременная маскировка

Любые два звука при одновременном прослушивании оказывают влияние на восприятие относительной громкости между ними. Более громкий звук снижает восприятие более слабого, вплоть до исчезновения его слышимости. Чем ближе частота маскируемого звука к частоте маскирующего, тем сильнее он будет скрываться. Эффект маскировки не одинаков при смещении маскируемого звука ниже или выше по частоте относительно маскирующего. Низкочастотный звук маскирует высокочастотные. При этом важно отметить, что высокочастотные звуки не могут маскировать низкочастотный.

### Временная маскировка

Это явление похоже на частотную маскировку, но здесь происходит маскировка во времени. При прекращении подачи маскирующего звука маскируемый некоторое время продолжает быть неслышимым. В обычных условиях эффект от временной маскировки длится значительно меньше. Время маскировки зависит от частоты и амплитуды сигнала и может достигать 100 мс.

В случае, когда маскирующий тон появляется по времени позже маскируемого, эффект называют пост-маскировкой. Когда маскирующий тон появляется раньше маскируемого (возможен и такой случай), эффект называют пре-маскировкой.

### Постстимульное утомление

Нередко после воздействия громких звуков высокой интенсивности у человека резко снижается слуховая чувствительность. Восстановление обычных порогов может продолжаться до 16 часов. Этот процесс называется «временный сдвиг порога слуховой чувствительности» или «постстимульное утомление». Сдвиг порога начинает появляться при уровне звукового давления выше 75 дБ и соответственно увеличивается при повышении уровня сигнала. Причём наибольшее влияние на сдвиг порога чувствительности оказывают высокочастотные составляющие сигнала.

Типы битрейта MP3

CBR расшифровывается как Constant Bit Rate, то есть постоянный битрейт, который задаётся пользователем и не изменяется при кодировании произведения. Таким образом, каждой секунде произведения соответствует одинаковое количество закодированных бит данных (даже при кодировании тишины).

VBR расшифровывается как Variable Bit Rate, то есть изменяющийся битрейт или переменный битрейт, который динамически изменяется программой-кодером при кодировании в зависимости от насыщенности кодируемого аудиоматериала и установленного пользователем качества кодирования (например, тишина закодируется с минимальным битрейтом). Минусом данного метода кодирования является то, что VBR считает «незначительной» звуковой информацией более тихие фрагменты, таким образом получается, что если слушать очень громко, то эти фрагменты будут некачественными, в то время как CBR делает с одинаковым битрейтом и тихие, и громкие фрагменты.

ABR расшифровывается как Average Bit Rate, то есть усредненный битрейт, который является гибридом VBR и CBR: битрейт в кбит/c задаётся пользователем, а программа варьирует его, постоянно подгоняя под заданный битрейт. Таким образом, кодек будет с осторожностью использовать максимально и минимально возможные значения битрейта, так как рискует не вписаться в заданный пользователем битрейт. Это является явным минусом данного метода, так как сказывается на качестве выходного файла, которое будет немного лучше, чем при использовании CBR, но хуже, чем при использовании VBR (при том же размере файла) .

**3 Практическая часть**

Сжатеи методом PCA яляется сжатием с потерями.

На рисунке 1 показан результат сжатия.

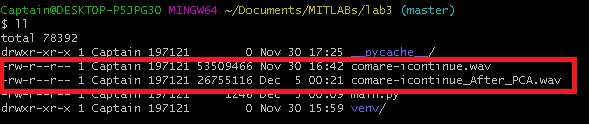


Рисунок 1 – Результат сжатия

Как видно из риснунка размер файла уменьшился в 2 раза. Сжатие производилось до 200 коломн.

**Вывод**

В ходе выполнение лабораторной работы были получены знания о Wav файлах, методах сжатия звука и методе главных компонент.

Было реализовано сжатие Wav файла на языке Python. По результатам сжатия можно сказать, что размер файл уменьшился в 2 раза, а звучание не сильно изменилось.

**Листинг**

from scipy.io import wavfile  
from sklearn.decomposition import PCA  
import numpy as np  
  
samplerate, audio\_file = wavfile.read('comare-icontinue.wav') # cчитывание музыки в формаnt wav  
signal\_wav = audio\_file[:, 0] #заполнения считанного сигнала нулями  
  
  
def pca\_reduce(signal, n\_components, block\_size=1024):  
 # дополнение сигнал нулями, для того,чтобы он делился на block\_size  
 samples = len(signal)  
 hanging = block\_size - np.mod(samples, block\_size) #остаток от деления  
 padded = np.lib.pad(signal, (0, hanging), 'constant', constant\_values=0) #заполения нулями  
 # изменение формы сигнала на размерность 1024  
 reshaped = padded.reshape((len(padded) // block\_size, block\_size))  
 # Метод главных компонент  
 pca = PCA(n\_components=n\_components)  
 pca.fit(reshaped)  
 transformed = pca.transform(reshaped)  
 reconstructed = pca.inverse\_transform(transformed).reshape((len(padded)))  
 return pca, transformed, reconstructed  
  
  
reconstructed = pca\_reduce(signal\_wav, 250, 1024)  
wavfile.write("comare-icontinue\_After\_PCA.wav", samplerate,  
 reconstructed[2].astype(np.int16)) # запись полученного результата