Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
«Высшая школа интеллектуальных систем и суперкомпьютерных технологий»

КУРСОВОЙ ПРОЕКТ

РАСПОЗНАВАНИЕ ДВУХТОНАЛЬНОГО МНОГОЧАСТОТНОГО НАБОРА ТЕЛЕФОННОГО НОМЕРА

по дисциплине «Методы обработки экспериментальных данных»

Выполнил
студент гр. 3540203/10101 В.В. Сухомлинов

Руководитель
доцент ВШИСиСТ, к.ф.-м.н. И.Н. Белых

Санкт-Петербург
2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Двухтональный многочастотный набор (DTMF) — это метод представления цифр клавиатуры телефона тонами для передачи по аналоговому каналу связи. Технология DTMF представляет собой надежную альтернативу роторным телефонным системам и позволяет пользователю вводить данные во время телефонного разговора. Эта функция позволила создать интерактивные системы автоматического ответа, такие как системы, используемые для телефонного банкинга, маршрутизации звонков в службу поддержки клиентов, голосовой почты и других подобных приложений.

Частоты, выбранные для тонов DTMF, имеют некоторые отличительные характеристики и уникальные свойства [1]:

- все тона находятся в слышимом диапазоне частот, что позволяет человеку определить, когда была нажата клавиша;
- ни одна частота не является кратной другой;
- сумма или разность любых двух частот не равна другой выбранной частоте.

Второе и третье свойства упрощают декодирование DTMF и уменьшают количество ложно распознанных тонов. Уникальные свойства позволяют приемникам DTMF определять, когда пользователь нажимает несколько клавиш одновременно.

Цель данной работы заключается в создании инструмента для генерации DTMF-сигналов и распознавания их в звуковом файле соответственно.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить теоретические материалы по моделированию DTMF-сигналов;
- реализовать метод генерации двухтонального многочастотного сигнала;
- изучить материалы по распознаванию DTMF-сигналов;
- выбрать и реализовать один из методов декодирования.

В результате данной работы предполагается создание программного инструмента, который способен как моделировать DTMF-сигналы из входящего набора символов, так и декодировать звуковую дорожку в текстовое сообщение.

ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Формат входного сигнала

В качестве входного сигнала был выбран аудиоформат wav по следующим причинам:

– отсутствие сжатия данных;

– наличие готовых инструментов для чтения/записи.

1.2. Алгоритм генерации сигнала

DTMF-сигнал представляет собой аддитивную модель двух гармонических процессов [1]:

$$x_k(t) = A_0 * sin(2 * \pi * f_1 * k * \triangle t) + A_0 * sin(2 * \pi * f_2 * k * \triangle t), \qquad (1.1)$$

где $k = 0,1,...,N-1$, A_0 - амплитуда сигнала, f_1 и f_2 - частоты гармоник, $\triangle t$ - шаг дискретизации.

Частоты гармоник берутся по приведённой ниже табл.1.1 из столбца и строки, соответствующих передаваемому символу. Каждая строка набора представлена частотой низкого тона, а каждый столбец - частотой высокого тона.

Таблица 1.1

Таблица соответствия частот и символов DTMF [1]

1209 Гц	**1336 Гц**	**1477 Гц**	**1633 Гц**	
1	2	3	A	**697 Гц**
4	5	6	B	**770 Гц**
7	8	9	C	**852 Гц**
*	0	#	D	**941 Гц**

Шаг дискретизации определяется, как отношение единицы к частоте дискретизации (rate) [2]. Согласно стандарту, для DTMF-сигнала приемлемым значением rate является 8000 Гц [1]. Однако это не необходимость: можно выбрать и более высокую частоту дискретизации - в конечном итоге это влияет больше на время вычислений (кодирования декодирования сигнала). Чтобы уменьшить время подсчета частот далее воспользуемся рекомендациями стандарта и будем использовать частоту дискретизации - 8000 Гц.

Сам процесс генерации сообщения двухтонального многочастотного набора заключается в последовательной генерации множества значений гармоник для каждого символа сообщения с последующей записью в файл.

1.3. Декодирование сигнала DTMF

Опишем базовый алгоритм декодирования сигнала. Представим, что мы записали в wav-файл звук символов ”3 ∗ 33”, как показано на рис.1.1.

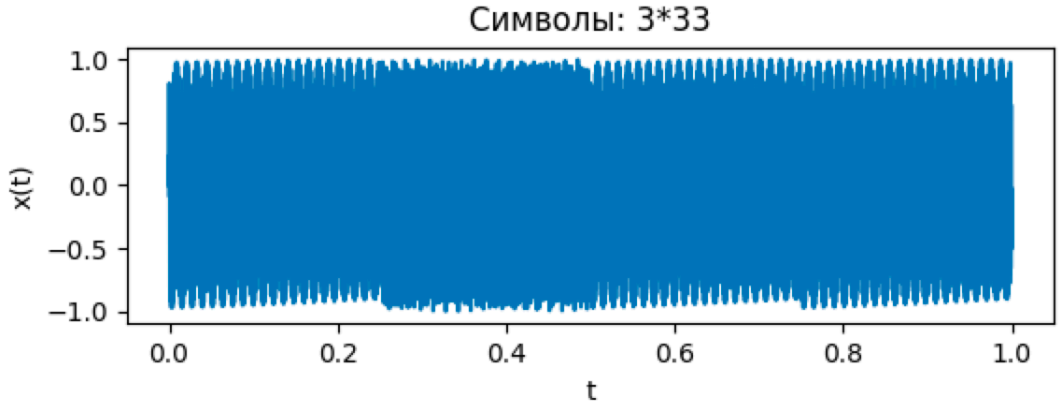

Рис.1.1. График сигнала ”3 ∗ 33”

С помощью спектра Фурье на рис.1.2 мы можем увидеть набор частот, используемых в нашем сообщении, и даже то, как часто они повторяются исходя из амплитуды определенных частот:

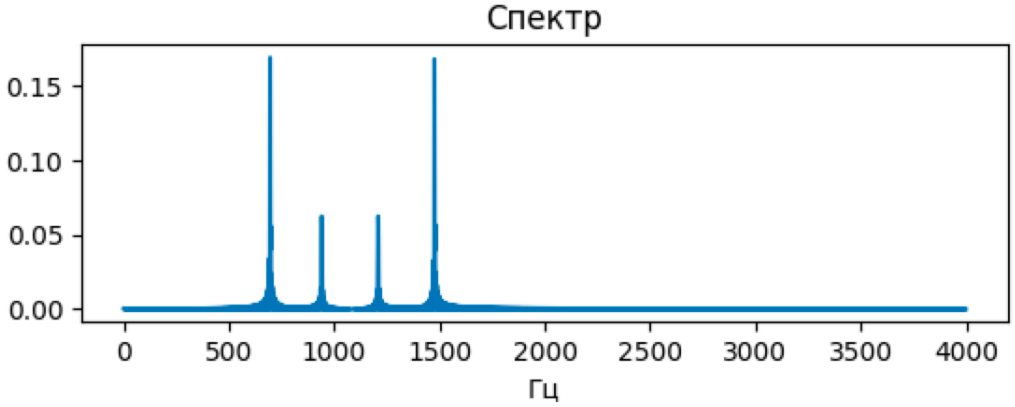

Рис.1.2. Спектр сигнала ”3 ∗ 33”

Минусом подобного решения в лоб является то, что мы не можем определить порядок символов в сообщении. Чтобы это исправить, можно обрабатывать сигнал пачками - мы заранее знаем, сколько секунд длится каждый сигнал, поэто-

му нам не составит труда для каждого отрезка построить спектр и извлечь из него необходимые частоты.

На примере выше, каждый сигнал длится по 0,25 секунд, поэтому размер одной такой пачки обработки будет равен произведению времени одного сигнала на частоту дискретизации. Вот полученные значения второго символа на рис.1.3:

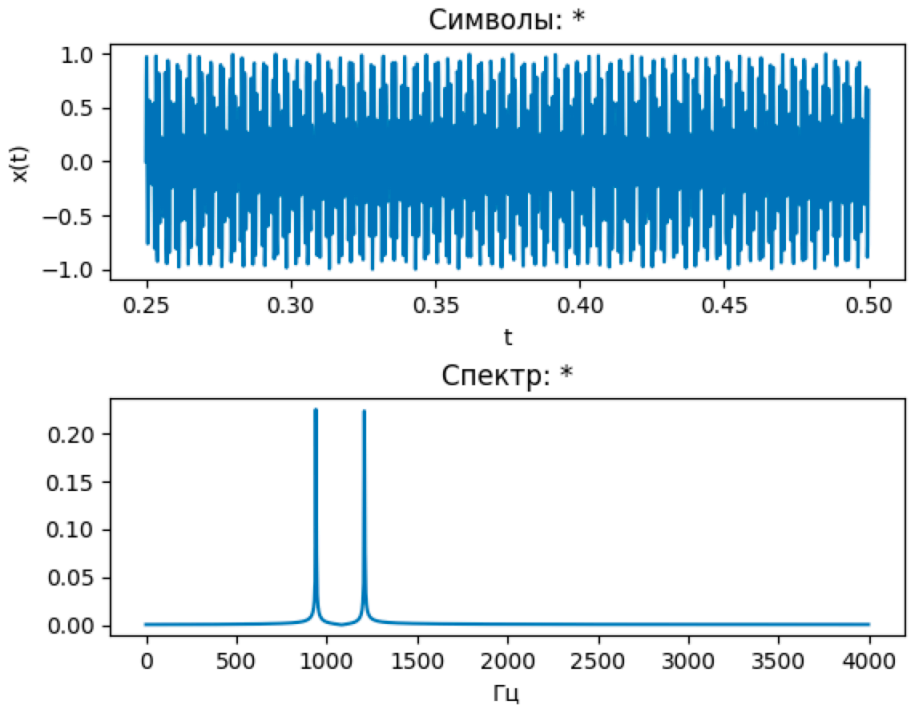

Рис.1.3. Спектр сигнала ” ∗ ”

Недостатком примитивного решения является - скорость. К сожалению, подсчет спектра для каждой пачки значений достаточно дорогостоящая по времени операция, особенно при увеличении частоты дискретизации. Поэтому воспользуемся специальными алгоритмами преобразования Фурье.

Для решения задачи детектирования и декодирования тональных сигналов в телефонии обычно применяются две вариации дискретного преобразования Фурье: быстрое преобразование Фурье (FFT) и алгоритм Гёрцеля.

В рамках данной работы воспользуемся последней, так как в отличие от быстрого преобразования Фурье, вычисляющего все частотные компоненты ДПФ, нам уже заранее известны частотные компоненты, которые мы хотим найти.

Алгоритм Гёрцеля заключается в следующем. Пусть x_n, $n = 0, \ldots, N-1$ — измеренные значения сигнала, которые являются входными данными для дискретного преобразования Фурье, а X_k, $k = 0, \ldots, N-1$ — частотные компоненты

дискретного преобразования Фурье, так как нам не важны их фазы будем искать магнитуды X_k^2.

Для расчёта X_k^2 с помощью алгоритма Гёрцеля последовательно вычисляются члены последовательности s_n для $n = 0, \ldots, N-1$ по рекуррентной формуле:

$$s_n = 2\cos\left(\frac{2\pi k}{N}\right) s_{n-1} - s_{n-2} + x_n, [1] \tag{1.2}$$

где $s_{-1} = s_{-2} = 0$, $k_n = [0.5 + \frac{n*N}{rate}]$.

Искомое значение получается как:

$$X_k^2 = s_{N-1}^2 - 2\cos\left(\frac{2\pi k}{N}\right) s_{N-1}s_{N-2} + s_{N-2}^2. \tag{1.3}$$

Найденные частоты сопоставляются с частотной табл.1.1.

Алгоритм Гёрцеля позволяет эффективно работать с достаточно высокими частотами дискретизации, например, 22050 или 44100.

1.4. Выводы

В рамках данной главы был выбран аудио-формат wav для работы со сгенерированными звуковыми сигналами.

Рассмотрели способ кодирования сообщений в соответсвии с приведенной табл.1.1, а также подобрали частоту дискретизации для эффективного, с прпрактической точки зрения, кодирования сообщения.

Для декодирования исследовали три варианта поиска частотных компонент: дискретное преобразование Фурье, быстрое преобразование Фурье и алгоритм Гёрцеля. Как итог, выбрали последний, так как он требует меньшего количества операций для расчета частотных компонент.

Рассмотрим практическую реализацию алгоритмов кодирования и декодирования DTMF-сигналов в следующей главе.

ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Используемые инструменты

Для реализации был выбран язык программирования Python 3.9. Среда разработки: JetBrains PyCharm. Также, использовались такие пакеты для языка Python, как:

– math, numpy - для расчетов;

– scipy - для работы с wav-файлом.

Для работы с гармоническими процессами использовались ранее реализованные в рамках курса [2] функции библиотеки spbstu-processing-data.

2.2. Модели и константы

Для работы с данными генерируемых сигналов был создан класс Signal, который хранит в себе значения сигнала, частоту дискретизации и интерпретируемый символ.

Значения частот, соответствующих символов приведены в программе в виде констант:

– DTMF_TABLE - словарь символов и частот;

– DTMF_FREQ - массив возможных частот набора;

– DTMF_HIGH - массив высоких частот;

– DTMF_LOW - массив низких частот.

2.3. Чтение и запись

Чтобы записывать значения сигналов в файл был создан класс Writer, который в функции *def write(filename: str, signals: [Signal])* формирует из объектов Signal весь массив значений и передает его на вход функции *write* библиотеки scipy.

Для чтения данных из wav-файла используется класс Reader и функция *def read(filename: str)*, которая обращается к *read* фреймворка scipy.

2.4. Генерация сигнала

Чтобы создать звуковой файл, был написан класс Generator, в котором реализованы две функции:

- *def generate_from(symbols: str, duration, volume, rate) → [Signal]* - принимает на вход строку символов с заданными параметрами продолжительности, громкости и частоты и возвращает массив сгенерированных элементов Signal;
- *def calculate(symbol: str, duration, volume, rate) → Signal* - для входного символа вычисляет значение двух гармоник и их аддитивную модель.

Результат функции *generate_from* передается объекту класса Writer, описанному ранее.

2.5. Распознавание символов

Алгоритм Гёрцеля реализован в рамках класса Goertzel, в котором используются следующие функции и методы:

- *init* - инициализатор класса, в котором заранее подсчитываются значения коэффициентов DTMF-частот;
- *def calc_s_n(self, sample_data)* - вычисляет значения последовательности s_n;
- *def calc_power(self) -> {float: float}* - вычисляет мощность для каждого частотного компонента;
- *def get_number(self, powers)* - на основе полученных магнитуд находим необходимый нам символ по таблице DTMF_TABLE;
- *def reset(self)* - для каждого последующего пакета значений сигнала сбрасываем посчитанные значения последовательности s_n.

Чтобы определить символы, которые были закодированы в wav-файле, был определен класс Detector. Он включает в себя одну функцию:

- *def detect(rate, data) -> str* - принимает на вход частоту и значения сигнала, а возвращает строку с распознанным сообщением.

В рамках этапа распознавания разбиваем массив значений сигнала на пакеты (bins), элементы которых поочередно передаем на вход алгоритма Гёрцеля - объекту класса Goertzel. В итоге получаем строку распознанных значений.

2.6. Пример работы генерации сигнала

Результатом работы является wav-файл, который можно скачать по ссылке - https://suhomlinov.com/sound.wav. Спектрограмма сигнала для сообщения ”147 ∗ ” на рис.2.1.

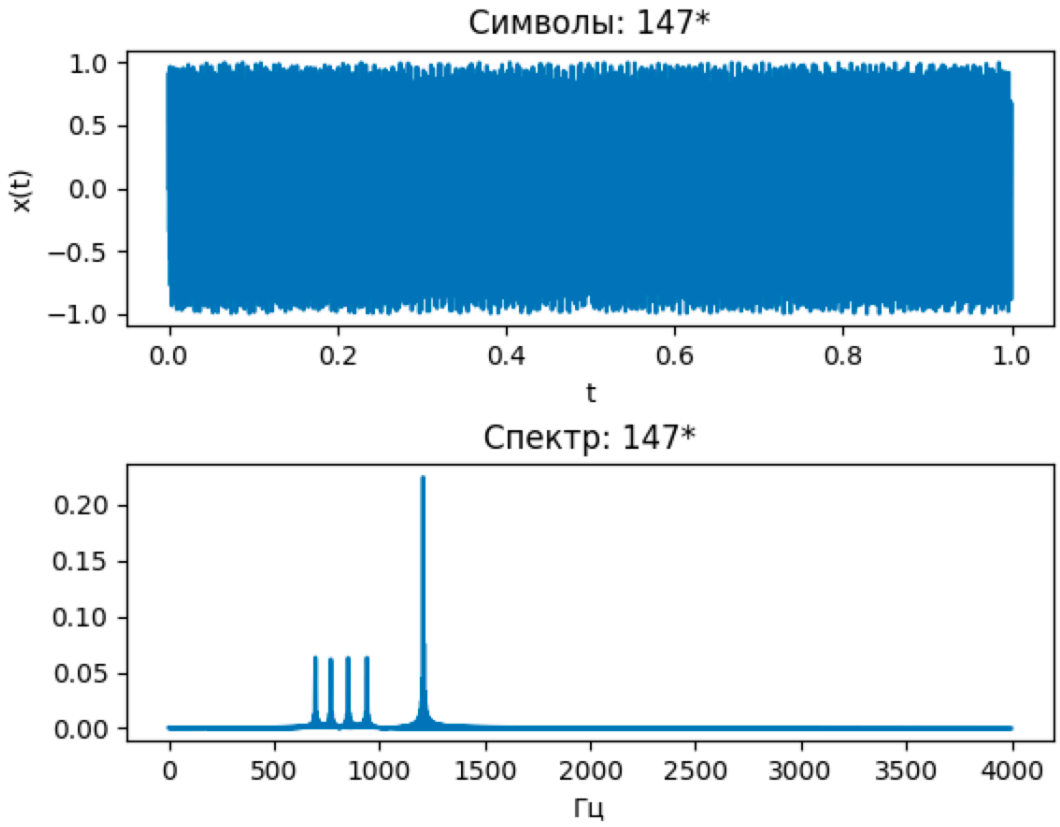

Рис.2.1. Спектрограмма сигнала сообщения ”147 ∗ ”

2.7. Пример работы декодирования сигнала

Результатом работы детектирования является строка с распознанным текстом, который можно удобно распечатать, как показано на

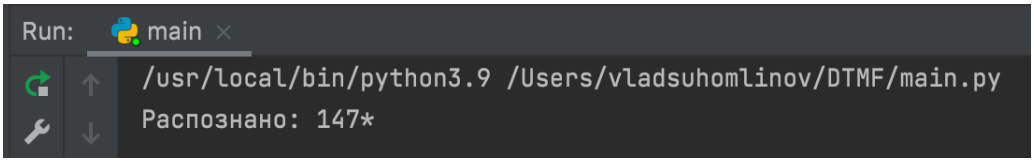

Рис.2.2. Вывод сообщения о распознанном сигнале

Теперь необходимо обратимо обратиться к проблеме шума. Не исключено, что при передаче аудио-сигнал может искажаться случайным шумом.

При небольшом шуме, превышающий исходный сигнал по амплитуде исходный в 2 раза, для сообщения ”147 ∗ ” мы можем уверенно распознать необходимые нам частоты благодаря спектру Фурье [3], как показано на рис.2.3:

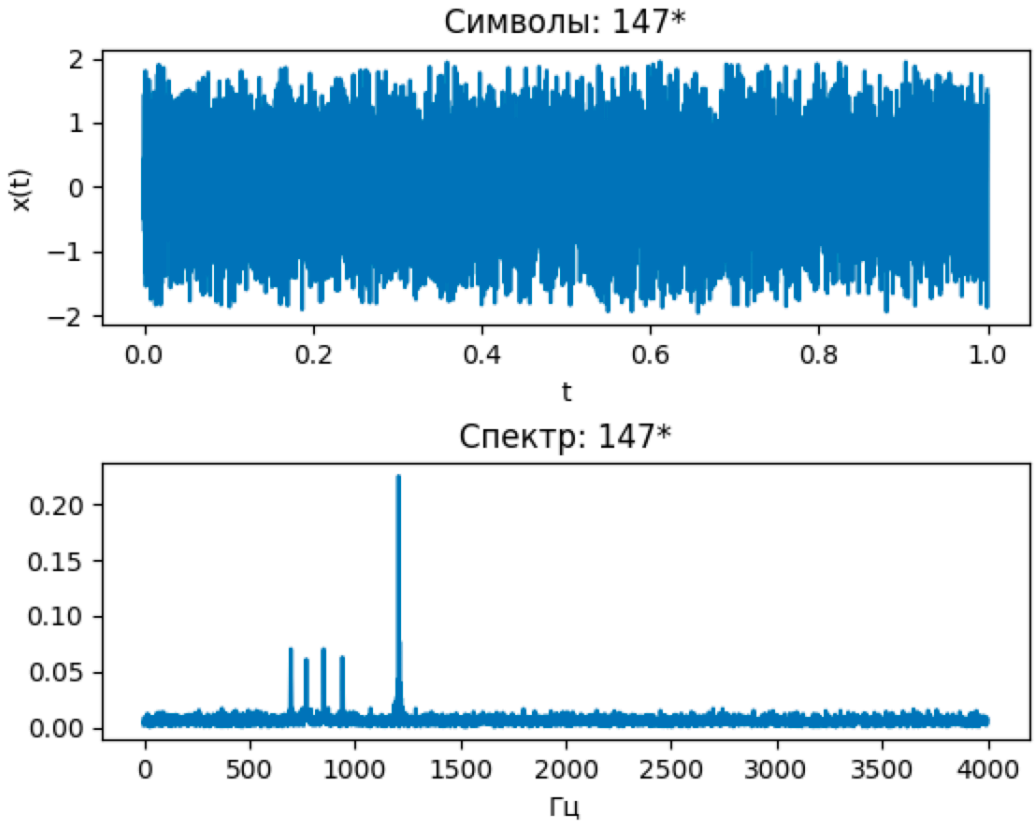

Рис.2.3. Результат применения шума к сигналу

Аналогичные эксперименты показали, что применение спектра Фурье достаточно даже для ситуаций, когда шум превышает по амплитуде полезный сигнал в 4 раза.

2.8. Выводы

В результате работы удалось реализовать инструменты для работы с звуковыми данными:

- реализован класс для чтения звукового файла и записи данных в него;
- написан инструмент для генерации DTMF-сигналов;
- успешно закодирован алгоритм Гёрцеля для распознавания частотных компонент;
- проверена работа распознавания полезного сигнала на фоне шума и без помех.

ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом были выполнены следующие задачи:

– изучены теоретические материалы по моделированию DTMF-сигналов;
– реализован метод генерации двухтонального многочастотного сигнала;
– изучены материалы по распознаванию DTMF-сигналов;
– выбран и реализован методов декодирования звукового сигнала.

Можно подвести вывод, что процесс генерации исследуемых сигналов не отличается вариативностью и использовать что-либо иное отличное от простого сложения двух гармоник нецелесообразно. Разработанная модель позволяет записывать сообщения как с низкой частотой дискретизации, так и с довольно-таки высокой от 44100 и более, что является успехом.

Процесс декодирования DTMF-сигнала наоборот предлагает множество возможных решений. Выбранный алгоритм декодирования - алгоритм Гёрцеля - оказался эффективным и мощным инструментом для быстрого выделения необходимых частотных компонент в сообщении с низкой степенью ошибки даже при наличии шума. Таким образом, применение спектра Фурье для распознавания в специальной реализации показал себя с лучшей стороны.

Разработанное решение имеет пути дальнейшего развития и улучшения, например, можно декодирование сигнала можно обрабатывать меньшими пакетами для увеличения точности распознавания и уменьшения влияния шума.

Как итог, можно считать, что поставленная цель достигнута - создан инструмент для генерации DTMF-сигналов и распознавания их в звуковом файле соответственно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DTMF DECODER REFERENCE DESIGN. — URL: https://www.silabs.com/documents/public/application-notes/an218.pdf (дата обращения: 16.10.2021).

2. *Белых И.* Лекции по дисциплине «Методы обработки экспериментальных данных». — 2021.

3. *Бендат Д.*, *Пирсол А.* Прикладной анализ случайных данных. — Мир, 1989. — 540 с.

Приложение 1

Исходный код

```
# Таблицы частот для кодирования и декодирования

DTMF_TABLE = {
    '1': [1209, 697],
    '2': [1336, 697],
    '3': [1477, 697],
    'A': [1633, 697],

    '4': [1209, 770],
    '5': [1336, 770],
    '6': [1477, 770],
    'B': [1633, 770],

    '7': [1209, 852],
    '8': [1336, 852],
    '9': [1477, 852],
    'C': [1633, 852],

    '*': [1209, 941],
    '0': [1336, 941],
    '#': [1477, 941],
    'D': [1633, 941],
}

DTMF_FREQ = [1209.0, 1336.0, 1477.0, 1633.0, 697.0, 770.0,
   852.0, 941.0]
DTMF_HIGH = [1209.0, 1336.0, 1477.0, 1633.0]
DTMF_LOW = [697.0, 770.0, 852.0, 941.0]

# -----------------
# Класс Reader
# -----------------

class Reader:

    @staticmethod
    def read(filename: str):
        # Читает wav файл.
        #
```

```
        # Parameters
        # ----------
        # filename : путь к файлу.
        #
        # Returns
        # -------
        # rate : int
        #     Частота дискретизации.
        # data : numpy array
        #     Данные файла.

        return scipy.io.wavfile.read(filename)

# -----------------
# Класс Writer
# -----------------

class Writer:

    @staticmethod
    def write(filename: str, signals: [Signal]):
        # Записывает данные в wav файл.
        #
        # Parameters
        # ----------
        # filename : путь к файлу.
        # signals : массив закодированных DTMFсигналов-.

        rate = 0
        data = np.array([])

        for signal in signals:
            rate = signal.rate
            data = np.append(data, signal.np_y_array())

        scipy.io.wavfile.write(filename, rate, data)

# -----------------
# Класс Signal
# -----------------

 class Signal:
    # Класс определеяет один DTMFсигнал-.
    #
    # Properties
```

```
        # ----------
        # data : путь к файлу.
        # symbol : символ DTMFсигнала-.
        # rate : частота дискретизации.

        # Initialization
        def __init__(self, data, symbol, rate):
            self.y_array = data[1]
            self.symbol = symbol
            self.rate = rate

        def np_y_array(self):
            # Преобразует данные в numpy array.
            #
            # Returns
            # -------
            # data : numpy array
            #     Данные файла.

            return np.array(self.y_array)

# -----------------
# Класс Detector
# -----------------

class Detector:

    @staticmethod
    def detect(rate, data) -> str:
        # Декодирует wav файл.
        #
        # Parameters
        # ----------
        # rate : путь к файлу.
        # data : данные сигнала.
        #
        # Returns
        # -------
        # result : str
        #     декодированная строка сигнала.

        result = ""
        bin_size = int(rate * .25)
        goertzel = Goertzel(rate, bin_size)

```

```
        for i in range(0, len(data) - bin_size + 1, bin_size):
            goertzel.reset()

            for j in range(bin_size):
                goertzel.calc_s_n(data[i + j])

            powers = goertzel.calc_power()
            symbol = goertzel.get_number(powers)

            result += symbol

        return result

# -----------------
# Класс Generator
# -----------------

class Generator:

    @staticmethod
    def generate_from(symbols: str, duration=.25, volume=.25,
  rate=8000) -> [Signal]:
        # Кодирует строку в сигнал.
        #
        # Parameters
        # ----------
        # symbols : строка символов.
        # duration : длительность одного сигнала.
        # volume : громкость.
        # rate : частота дискретизации.
        #
        # Returns
        # -------
        # generated : [Signal]
        #     массив DTMFсигналов-.

        generated = []

        for symbol in symbols:
            generated.append(Generator.calculate(symbol,
  duration, volume, rate))

        return generated

    @staticmethod
```

```
    def calculate(symbol: str, duration=.25, volume=.25, rate
=8000) -> Signal:
        # Кодирует символ в сигнал.
        #
        # Parameters
        # ----------
        # symbol : символ.
        # duration : длительность одного сигнала.
        # volume : громкость.
        # rate : частота дискретизации.
        #
        # Returns
        # -------
        # generated : Signal
        #     закодированный DTMFсигналов-.

        first_garmonik = GarmonikModel(volume, DTMF_TABLE[symbol
.upper()][0], 0, int(duration * rate), 1, 1 / rate) \
            .trend(0, None)
        second_garmonik = GarmonikModel(volume, DTMF_TABLE[
symbol.upper()][1], 0, int(duration * rate), 1, 1 / rate) \
            .trend(0, None)
        final_garmnonik = ModelDriver.add(first_garmonik,
second_garmonik)

        return Signal(final_garmnonik, symbol, rate)

# -----------------
# Класс Goertzel
# -----------------

class Goertzel:
    # Реализация алгоритма Герцеля.

    def __init__(self, sample_rate: int, bin_size: int):
        self.s_prev = {}
        self.s_prev2 = {}
        self.coeff = {}

        for k in DTMF_FREQ:
            self.s_prev[k] = .0
            self.s_prev2[k] = .0

            freq_k = .5 + (bin_size * k) / sample_rate
```

```
215             self.coeff[k] = 2.0 * math.cos(2.0 * math.pi *
    freq_k / bin_size)

     def get_number(self, powers):
         # Возвращает символ соответствующий
         # полученным частотных компонентам.
220          #
         # Parameters
         # ----------
         # powers : магнитуды частот.
         #
225          # Returns
         # -------
         # key : str
         #     декодированный DTMFсимвол-.

230          high_freq = .0
         high_freq_temp = .0
         low_freq = .0
         low_freq_temp = .0

235          for (high, low) in zip(DTMF_HIGH, DTMF_LOW):
             if powers[high] > high_freq_temp:
                 high_freq_temp = powers[high]
                 high_freq = high

240              if powers[low] > low_freq_temp:
                 low_freq_temp = powers[low]
                 low_freq = low

         for key in DTMF_TABLE:
245              if DTMF_TABLE[key][0] == high_freq and DTMF_TABLE[
    key][1] == low_freq:
                 return key

     def calc_s_n(self, sample_data):
         # Вычисляет Sn.
250          #
         # Parameters
         # ----------
         # sample_data : частота дикретизации.

255          for freq in DTMF_FREQ:
             s = self.coeff[freq] * self.s_prev[freq] - self.
    s_prev2[freq] + sample_data
```

```
            self.s_prev2[freq] = self.s_prev[freq]
            self.s_prev[freq] = s

    def calc_power(self) -> {float: float}:
        # Вычисляет магнитуды частот.
        #
        # Returns
        # -------
        # powers : {float: float}
        #     словарь частот и их магнитуд.

        powers = {}

        for freq in DTMF_FREQ:
            power = self.s_prev2[freq] ** 2 + self.s_prev[freq]
** 2 - \
                    self.coeff[freq] * self.s_prev[freq] * self.
s_prev2[freq]
            powers[freq] = power

        return powers

    def reset(self):
        # Удаляет ранее посчитанные данные.

        self.s_prev = {}
        self.s_prev2 = {}

        for k in DTMF_FREQ:
            self.s_prev[k] = .0
            self.s_prev2[k] = .0
```