Часто (но не всегда) информация о частоте основного тона в сигнале хранится в виде меток, которые обозначают границы периодов основного тона. Заметьте, что принципы расставления границ могут отличаться: у нас началом периода принято считать переход через 0 в положительную область, в других местах - минимальное значение (отрицательный пик).

I. Форматы записи меток периодов основного тона:

1. seg_G1 (Wave Assistant)

```
!wget https://pkholyavin.github.io/mastersprogramming/cta0001.seg_G1
```

```
from itertools import product
letters = "GBRY"
nums = "1234"
levels = [ch + num for num, ch in product(nums, letters)]
level_codes = [2 ** i for i in range(len(levels))]
code_to_level = {i: j for i, j in zip(level_codes, levels)}
level_to_code = {j: i for i, j in zip(level_codes, levels)}


def read_seg(filename: str, encoding: str = "utf-8-sig") -> tuple[dict, list[di
    with open(filename, encoding=encoding) as f:
        lines = [line.strip() for line in f.readlines()]

    # найдём границы секций в списке строк:
    header_start = lines.index("[PARAMETERS]") + 1
    data_start = lines.index("[LABELS]") + 1

    # прочитаем параметры
    params = {}
    for line in lines[header_start:data_start - 1]:
        key, value = line.split("=")
        params[key] = int(value)

    # прочитаем метки
    labels = []
    for line in lines[data_start:]:
        # если в строке нет запятых, значит, это не метка и метки закончились
        if line.count(",") < 2:
            break
        pos, level, name = line.split(",", maxsplit=2)
        label = {
            "position": int(pos) // params["BYTE_PER_SAMPLE"] // params["N_CHAN
            "level": code_to_level[int(level)],
            "name": name
        }
```

```
            labels.append(label)
        return params, labels


params, labels = read_seg("cta0001.seg_G1")
labels
```

На уровне G1 часто (но не всегда!) также находятся метки, обозначающие границы файла. Они не являются метками границ периодов ОТ!

```
pitch_labels = labels[1:-1]
```

2. PointProcess (Praat)

```
!wget https://pkholyavin.github.io/mastersprogramming/cta0001.PointProcess
```

```
with open("cta0001.PointProcess") as f:
    lines = f.readlines()
print(*lines, sep="")
```

Задание для выполнения в классе: напишите функцию, которая принимает на вход имя файла .PointProcess и возвращает список вещественных чисел, соответствующих позиции каждой метки.

Факультативное задание: напишите функцию, которая читает файл PointProcess и при этом не опирается на волшебные числа (7).

3. Файлы .pm (REAPER)

Установим утилиту REAPER:

```
%%bash
git clone https://github.com/google/REAPER.git
cd REAPER
mkdir build   # In the REAPER top-level directory
cd build
cmake ..
make
```

```
!wget https://pkholyavin.github.io/mastersprogramming/cta0001.wav
```

```
!REAPER/build/reaper -i cta0001.wav -p cta0001.pm -a
```

```
with open("cta0001.pm") as f:
    lines = f.readlines()
print(*lines, sep="")
```

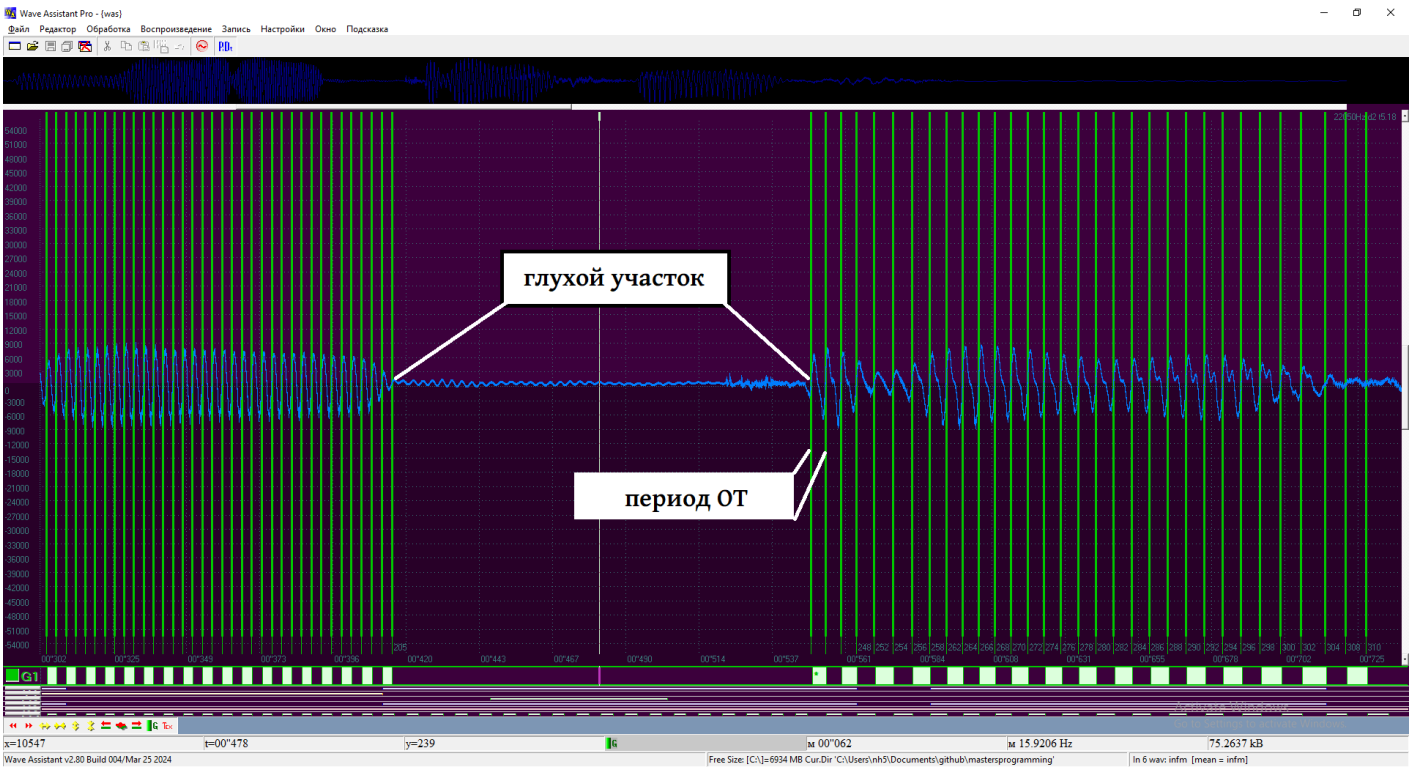
```
pitch_marks = [[float(line.split(" ")[0]), int(line.split(" ")[1])] for line in lines[7:]] #
print(pitch_marks)
```

II. Обработка меток периодов

Вопрос:

1. Как, зная время начала и конца периода, определить его длину?
2. Как, зная длину периода, определить значение частоты основного тона в этот момент времени?

Не любая пара меток в файле соответствует периоду ОТ. Некоторые из них отмечают глухие участки:



Отличить одно от другого мы можем, задав минимальное возможное значение ЧОТ. Тогда мы сможем отличить такие интервалы по длине: периодами ОТ будут только те интервалы,

которые короче определённого порога.

Вопрос: если минимальная возможная ЧОТ = 50 Гц, чему равна максимально возможная длина периода?

```
min_f0 = 50  # Hz
max_period = "чему?"
```

Задание для выполнения в классе: напишите функцию, которая принимает на вход имя файла .seg_G1 и пороговое значение ЧОТ (50 Гц по умолчанию) и возвращает два списка: в одном - положение каждого периода (т.е. его середина), во втором - значение ЧОТ в этом периоде или NaN, если это значение оказалось меньше порогового.

NaN (Not a Number) - значение, которое часто используют в качестве заместителя для пропущенных данных. Его можно получить, например, так:

```
nan_val = float("nan")
print(nan_val)
```

Или так:

```
import numpy as np
nan_val2 = np.nan
print(nan_val2)
```

NaN не равен никакому числу и не равен сам себе:

```
nan_val == nan_val
```

Чтобы проверить, является ли значение NaN, есть специальные функции:

```
np.isnan(nan_val)
```

Если мы будем строить графики по данным, содержащим NaN, с помощью matplotlib, то в соответствующих местах появятся разрывы. Сравните:

```
x = list(range(10))
y1 = [1, 2, 3, np.nan, np.nan, 5, 4, 3, 2, 1]
y2 = [1, 2, 3, 4, 5, 5, 4, 3, 2, 1]
```

```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(1, 2)
axes[0].plot(x, y1)
axes[1].plot(x, y2)
```

Вспомним, как можно обрабатывать метки попарно:

```
def print_label_pairs(filename):
    params, labels = read_seg(filename)
    for start, end in zip(labels, labels[1:]):
        print(start, end)
```

```
def get_f0(filename: str, min_f0: float = 50.0) -> tuple[list[float], list[float]]:
    times, f0_values = [], []
    # прочитать сег
    # убрать из него метки начала и конца файла
    # перебрать метки попарно
    # в каждом интервале:
    # определить место середины В СЕКУНДАХ, добавить в times
    # определить значение ЧОТ
    # если оно >= минимального, добавить в f0_values
    # в противном случае добавить в f0_values NaN
    return times, f0_values
```

Используем эту функцию для построения графика несглаженной ЧОТ:

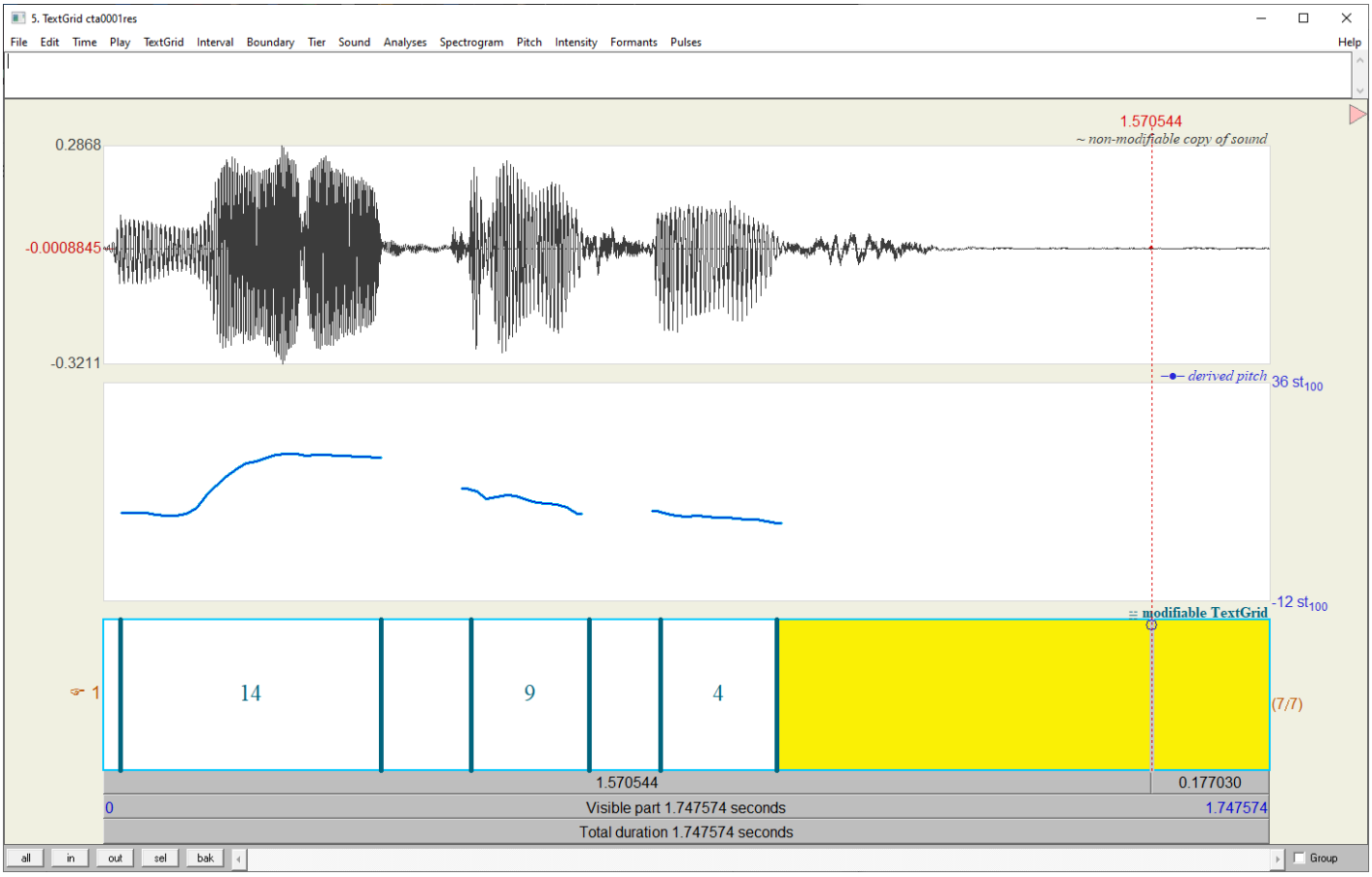
```
times, f0_values = get_f0("cta0001.seg_G1")
plt.plot(times, f0_values)
plt.show()
```

Домашнее задание: напишите программу, которая определяет границы каждого звонкого участка в файле по меткам ОТ и генерирует файл .TextGrid, содержащий интервалы, им соответствующие. Названием каждого интервала сделайте число, которое показывает разницу между максимальным и минимальным значением ЧОТ в этом участке.

Необязательное дополнение: выразите эту разницу в полутонах. Формула для перевода разницы между двумя частотами в полутона:

$$st = 12 \cdot \log_2 \frac{f_2}{f_1}$$

Что должно получиться:



Функции для логарифма:

```
from math import log2
log2(10)
```

```
import numpy as np
np.log2(10)
```

Возможный алгоритм:

```
def voiced_regions(seg_filename: str, min_f0: float = 50) -> None:
    # прочитаем метки из файла

    # создадим список, куда будем добавлять наши звонкие участки
    # каждый звонкий участок характеризуется тремя параметрами:
    # время начала, время конца, список значений ЧОТ внутри него
    # можно реализовать это в виде словаря с тремя ключами,
    # два из которых будут хранить числовые значения, третий - список чисел

    # сразу добавим в этот список первый участок
    # его время начала соотвествует первой метке
    # а время конца мы ещё не знаем
```

```
# переберём все пары меток
# если пара является настоящим периодом ОТ, то добавляем значение ЧОТ в текущий участок
# (т.е. последний участок в большом списке)
# иначе текущему участку назначаем время конца, равное левой метке в паре
# и добавляем в большой список новый участок со временем начала, равным правой метке в г

# после конца цикла последнему участку назначим время конца, равное последней метке

# создадим новый текстгрид, добавим в него интервальный уровень
# end_time этого уровня пусть будет равен времени самой последней метки в файле
# (которую мы не учитывали при анализе ЧОТ)

# теперь переберём все участки
# для каждого создадим новый интервал с границами, соответствующими границам участка
# вычислим перепад ЧОТ, переведём в строку, сделаем именем созданного интервала

# запишем текстгрид в файл
pass
```