Politechnika Warszawska
Wydział Matematyki i Nauk Informacyjnych

przedmiot
Analiza i Przetwarzanie Dźwięku, Projekt 1.

Aplikacja do analizy czasowej nagrań audio

Antoni Kingston
Numer albumu
327284

prowadzący
dr. inż. Janusz Rafałko

WARSZAWA marzec 2025

Spis treści

1. Opis ogólny

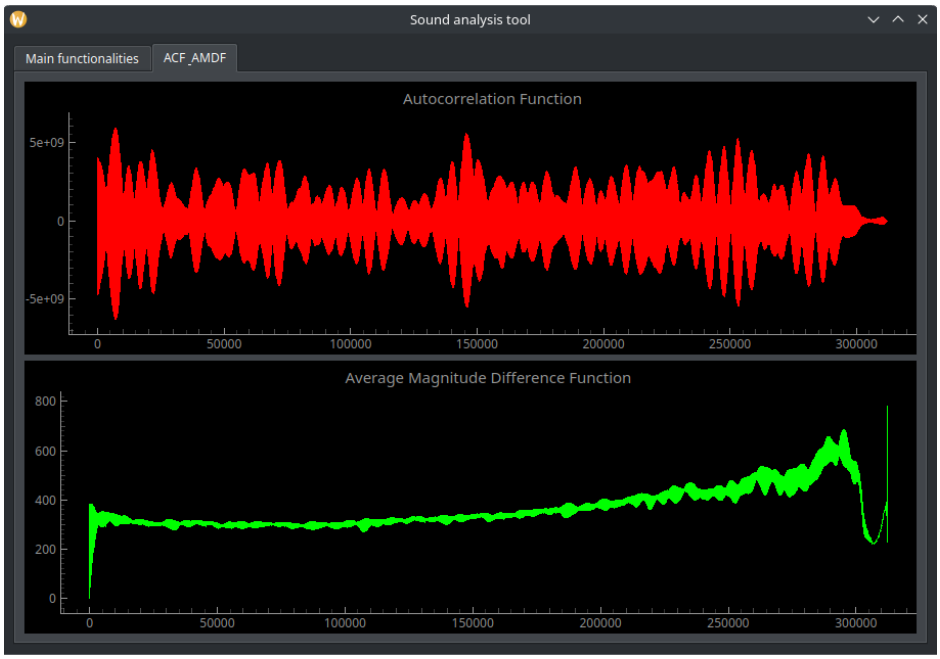
1.1. Użyte technologie

Aplikacja nosi nazwę „Sound analysis tool"i pozwala na zbadanie kilku podstawowych parametrów plików dźwiękowych. Została napisana w języku Python, ze względu na dostępność w nim zarówno narzędzi do analizy danych (numpy) jak i przystępnych bibliotek do tworzenia graficznego interfejsu (PyQt6). W jednym miejscu, celem poprawy wydajności, w drugim ze względu na lenistwo algorytmiczne autora, użyto funkcji z biblioteki scipy. Wszystkie potrzebne do uruchomienia aplikacji moduły wraz z wersjami znajdują się w pliku „requirements.txt"

1.2. Interfejs

Składa się on z dwóch zakładek. Na pierwszej, główny plan zajmuje pole na wykres przebiego czasowego, poniżej niego znajdują się przyciski do wczytania i odtworzenia pliku w formacie .wav, dalej dwa radioboxy, które jeśli aktywne, zaznaczają na wykresie odpowiednie fragmenty, a pod nimi dwa suwaki służące do określenia fragmentu podlegającego analizie. Z prawej strony zakładki zostały umieszczone pola z wartościami liczbowymi parametrów, które ulegają aktualizacji po naciśnięciu przycisku „Calculate Parameters". Na drugiej zakładce znajdują się 2 pola z wykresami autokorelacji i AMDF.

Rysunek 1.1. Pierwsza zakładka



Rysunek 1.2. Druga zakładka

2. Metody wyznaczania parametrów (wielkości) i ich implementacja

2.1. Format wczytywanych plików ich przechowywanie prze program

Aplikacja wczytuje pliki typu WAV, i przechowuje je jako pojedynczy jednowymiarowy wektor typu numpy. W przypadku, gdy plik jest dwukanałowy odrzucane są więc dane z 2. kanału. Poza tym przy wczytywaniu pobierane są metaparametry.

```
n_channels = wav_file.getnchannels()
self.sample_rate = wav_file.getframerate()
n_frames = wav_file.getnframes()
self.audio_duration = n_frames / self.sample_rate
self.audio_data = np.frombuffer(wav_file.readframes(n_frames), \
                                            dtype=np.int16)
self.audio_data = self.audio_data.astype(np.float32) / \
                                        np.iinfo(np.int16).max

if n_channels == 2:
    self.audio_data = self.audio_data[::2]
```

2.2. STE (Short Time Energy)

Uśrednione kwadraty próbki, obliczeniowo bliżej pierwotności niż głośność.

2.3. Volume (głośność)

Pierwiastek z STE. Wielkość bardziej interpretowalna fizycznie - jej logarytm jest mniej więcej (cokolwiek to oznacza) proporcjonalny do głośności odczuwanej przez człowieka.

```
    ste = np.mean(frame_data**2)
    volume = ste ** 0.5
```

2.4. ZCR

Jak sama nazwa wskazuje miara częstości przechodzenia danych przez poziom 0. Można ująć w następującą formułę (N - liczność ramki, n - numer ramki):

$$Z(n) = \frac{1}{2N}\left(\sum_{i=0}^{N-1} sgn\left(S_n(i)\right) - sgn\left(S_n(i-1)\right)\right)$$

```
def calculate_zcr(data):
    length = len(data)
    crosses = 0
    for i in range(1, length):
        if data[i - 1] * data[i] < 0:
            crosses += 1
    return crosses / (2 * length)
```

2.5. SR (silent ratio)

Wielkość wyliczana dla całych danych na podstawie obliczeń, dla poszczególnych ramek. Proporcja ramek zakwalifikowanych jako „ciche"w ich ogóle. Ramka klasyfikowana jest jako cisza jeśli głośność (STE) oraz ZCR są na wystarczająco niskim poziomie. W przypadku programu jest to 0.05 maksymalnej głośności spośród ramek oraz ZCR poniżej 0.03.

```
def calculate_silence_ratio_voiced_unvoiced(full_data, nframes = 256):
    ZCR_thresh = 0.03
    maxv = calculate_max_vol(full_data, nframes)
    silent = 0
    frames = np.array_split(full_data, nframes)
    silent_idxs =[]
    voiced_idxs = []
    unvoiced_idxs = []
    curr_idx = 0
    for frame in frames:
        vol = np.sqrt(sum(frame ** 2))
        if calculate_zcr(frame) < ZCR_thresh and vol < 0.05*maxv:
            silent += 1
            silent_idxs.append((curr_idx, curr_idx + len(frame)))
        elif calculate_zcr(frame) < ZCR_thresh and vol > 0.05*maxv:
            voiced_idxs.append((curr_idx, curr_idx + len(frame)))
        else:
```

```
            unvoiced_idxs.append((curr_idx, curr_idx + len(frame)))
        curr_idx += len(frame)
    return (silent/nframes), silent_idxs, voiced_idxs, unvoiced_idxs
```

2.6. Częstotliwość tonu podstawowego

Fragmenty dźwięczne są złożeniami drgań harmonicznych, ton podstawowy to jedno z tych drgań o najniższej częstotliwości. W aplikacji wyznaczane jest za pomocą funkcji AMDF (Average Magnitude Difference Fucntion). Jej piki oznaczają fragmenty raptownej zmiany w próbce. Przy założeniu (które niekoniecznie jest spełnione), że ton podstawowy przenosi najwięcej energii piki te odpowiadają przechodzeniu przez 0 (gdzie pochodne funkcji trygonometrycznych są najwyższe) oscylacji temu tonowi odpowiadających. Odległość między pikami jest wprost proporcjonalna do okresu F0, znając częstotliwość próbkowania jesteśmy w stanie wyznaczyć okres a zatem i częstotliwość wyznaczyć już dokładnie.

```
def calculate_f0(data, sample_rate):
    AMDF = amdf(data)
    amp = amplitude(AMDF)
    spacing = find_average_minima_spacing(AMDF, 0.85*amp)
    if not spacing:
        spacing = 1
    return sample_rate / spacing
```

Przy poszukiwaniu interesujących nas pików istotne jest zdefiniowanie wybitności powyżej, której chcemy szukać (by uniknąć ekstremów związanych z innymi tonami) po paru próbach zdecydowano się na 0.85 amplitudy. Funkcja *find_average_minima_spacing()* korzysta z funkcji *find_peaks()* z biblioteki scipy.

```
def find_average_minima_spacing(amdf_vals, prominence):
    minima_indices, _ = signal.find_peaks(-amdf_vals, prominence=prominence)
    if len(minima_indices) < 2:
        return None
    spacings = np.diff(minima_indices)
    avg_period = np.mean(spacings)
    return avg_period
```

2.7. AMDF

Wcześniej już wspominana. Interpretowalna jako lokalna miara monotonicznej zmienności próbki.

$$A_n(l) = \sum_{i=0}^{N-l-1} |s_n(i+l) - s_n(i)|$$

Wypisany wyżej wzór jeśli zaimplementowany wprost oznaczałby złożoność $o\left(n^2\right)$, jako że trwające znacznie mniej niż minutę nagrania przy powszechnie stosowanych częstotliwościach próbkowania przekraczają swoim rozmiarem 1000000 nie jest to porządane. Poniższa implementacja działa w czasie $o(nlogn)$ co zapobiega parudziesięciosekudnowemu zawieszaniu się aplikacji. (Na moment pisania sprawozdania autor nie zapoznał się ze szczegółami działania FFT, zobowiązuje się on raporcie za projekt 2. zdemaskować stosowaną przez siebie magię)

```
def amdf(data):
    N = len(data)
    signal_sq = data ** 2
    sum_x2 = np.cumsum(signal_sq[::-1])[::-1]
    sum_x2_shifted = np.roll(sum_x2, -1)
    fft_signal = np.fft.fft(data, 2 * N)
    auto_corr = np.fft.ifft(fft_signal * np.conj(fft_signal)).real[:N]
    amdf = (sum_x2[:N] + sum_x2_shifted[:N] - 2 * auto_corr) / N
    return amdf
```

2.8. ACF

Funkcja, której argumentem są przesunięcia a wartościami korelacje próbkowe sygnału ze swoją przesuniętą wersją.

$$R_n(l) = \sum_{i=0}^{N-l-1} s_n(i)s_n(i+l)$$

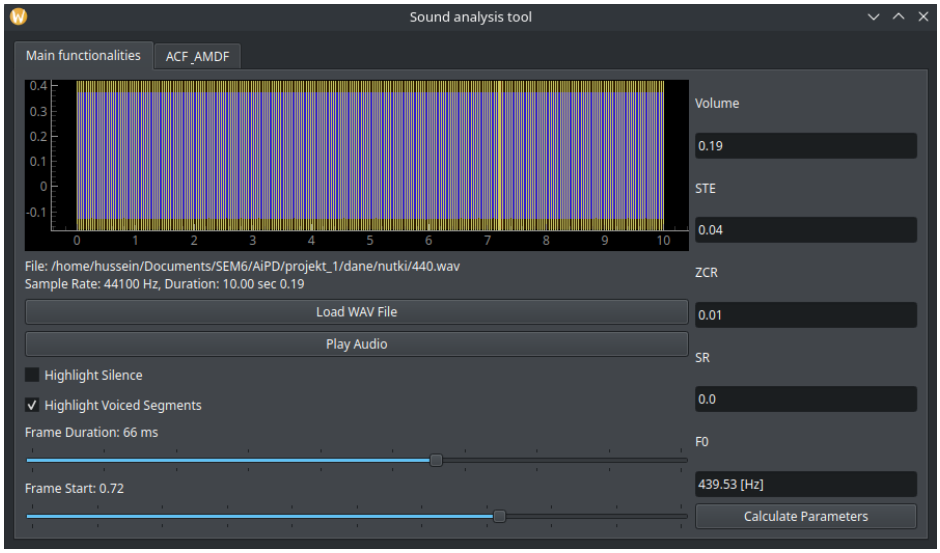
Ponownie naiwna implementacja daje niesatysfakcjonująca złożoność stosowana jest zatem FFT.

```
def acf(data):
    ACF = signal.fftconvolve(data, data, mode='full')
    ACF = ACF[len(ACF)//2:]
    return ACF
```

3. Prezentacja wyników działania

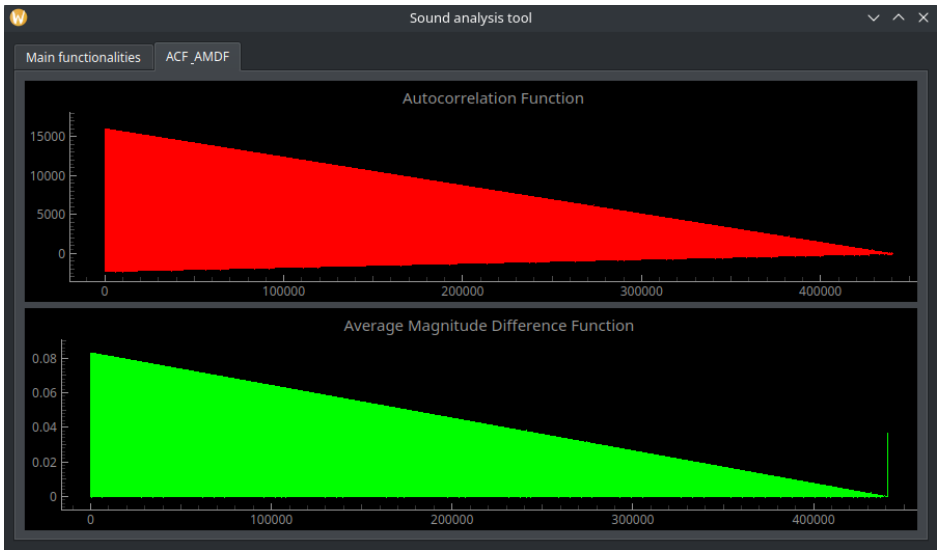
3.1. Czyste A4

Zbadano tutaj czysty sinusoidalny sygnał o częstotliwości 440Hz (autor z góry przeprasza zwolenników starego strojenia). Całe nagranie oczywiście jest dźwięczne (bardzon niskie ZCR) co udaje się programowi wykryć. Pojedynczy występujący ton określany jest z bardzo dobrą dokładnością. Ze względu na prostotę sygnału funkcja autokorelacji AMDF



Rysunek 3.1. Na żółto oznaczone są obszary dźwięczne

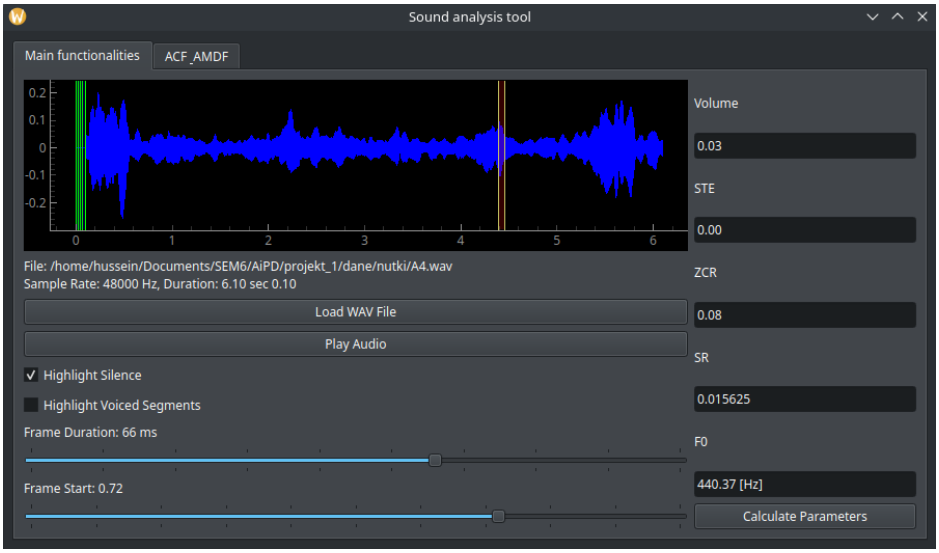
wyglądają bardzo regularnie.



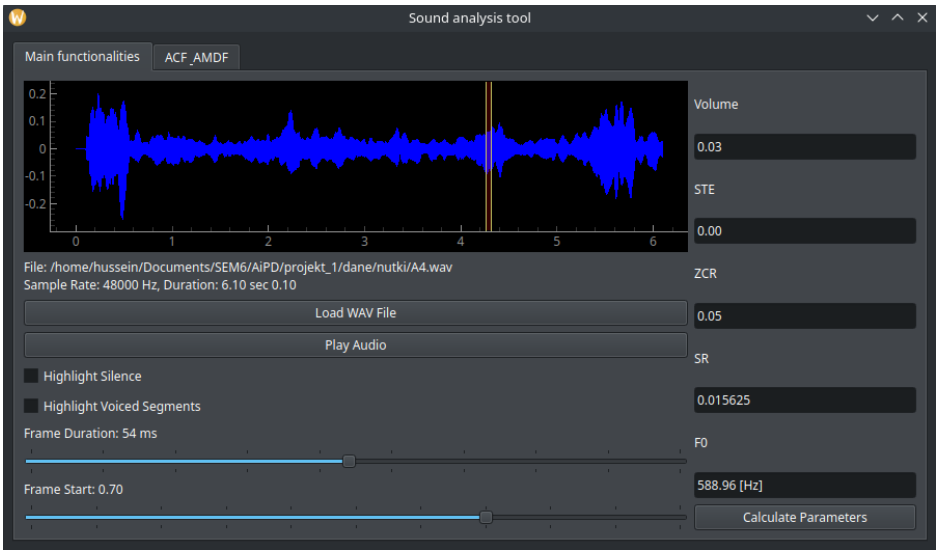
Rysunek 3.2. Zakładka 2.

3.2. Zaburzone A4

W tym przypadku nagranie pochodzi z mikrofonu, który nagrał dźwięk wydobywający się z telefonu z odpaloną aplikacją z pianinem. Przy pewnym zaznaczeniu obszaru udaje się uzyskać właściwą częstotliwość nie jest to jednak zdarzenie gwarantowane.



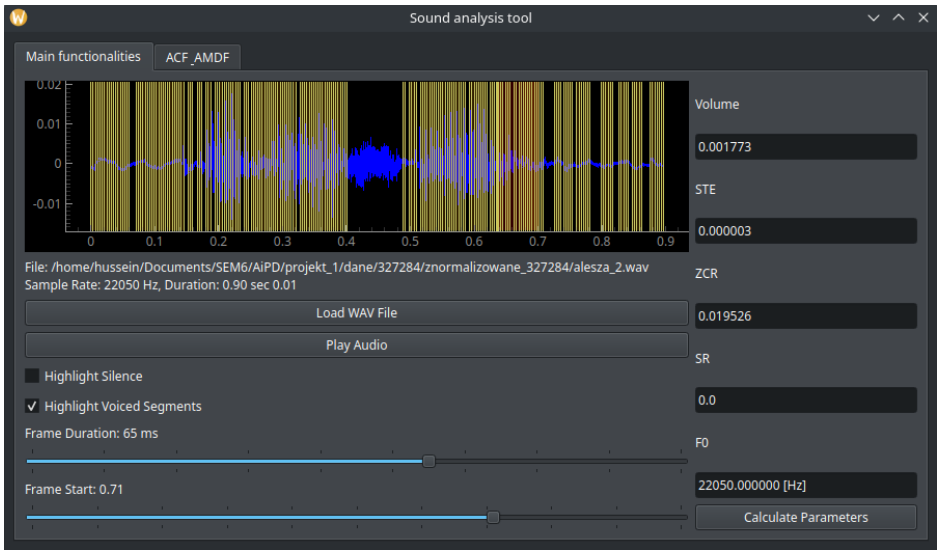
Rysunek 3.3. Na zielono oznaczono ciszę



Rysunek 3.4. Przy innym zaznaczeniu wyznaczona częstotliwość przestaje przystawać do oczekiwań

3.3. Alesza

Porównano dwie krótkie próbki nagarań pseudowyrazu „alesza"jedną głosu męskiego, drugą żeńskiego. W obydwu wypadkach udało się wykryć bezdźwięczną głoskę „sz".

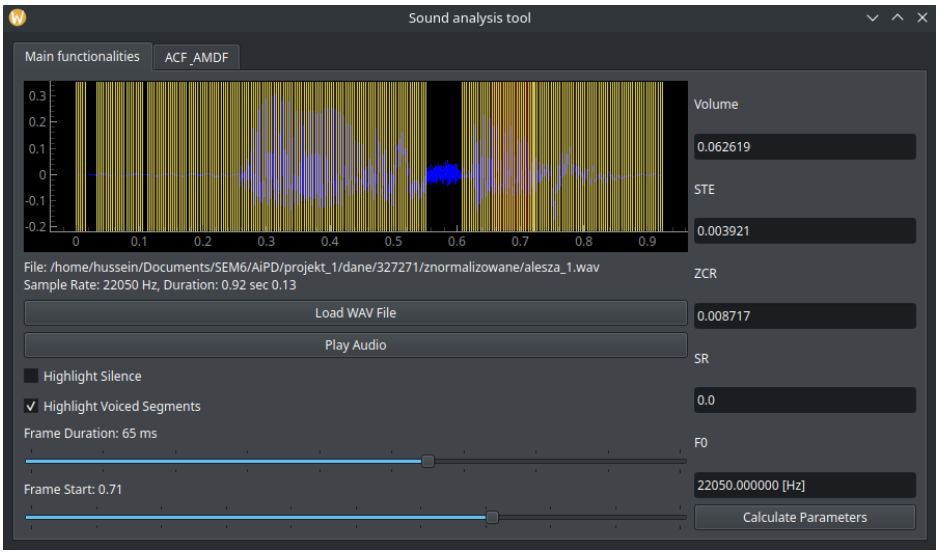


Rysunek 3.5. Głos męski

Trudno wśród wyznaczanych parametrów i wielkości znaleźć wyróżniki głosu męskiego i żeńskiego.

3.4. Głos + Muzyka

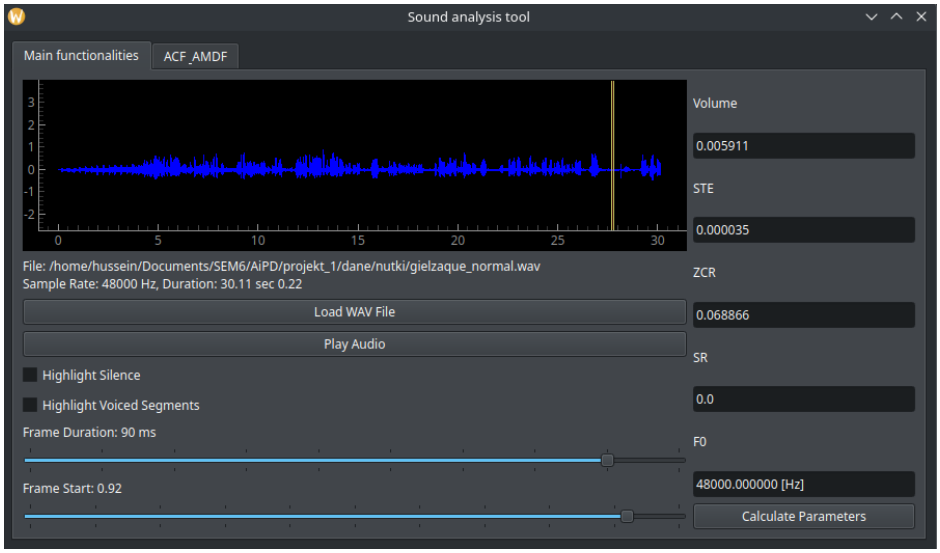
Na koniec przyjżano się nagraniu czołówki filmo z platformy „youtube", w którym ludzki głos miesza się z muzyką. Autor sprawozdania miał wyraźny problem z podaniem interesującej informacji z tym przyjżeniem się związanej. Niemniej jednak na obszarach o większej amplitudzie na wykresie obliczana głośność rzeczywiście jest większa.



Rysunek 3.6. Głos żeński

4. Wnioski

Z pewnością zaimplementowana metoda wyznaczania częstotliwości tonu podstawowego jest do poprawy, daje zadowalające efekty tylko w wyidealizowanej sytuacji. Wyliczanie pozostałych parametrów pozwala z satysfakcjonująca dokładnością wykrywać ciszę, obszary dźwięczne oraz bezdźwięczne. Interfejs posiada pewne braki z pewnością przydałaby się opcja zaznaczania obszaru pracy za pomocą myszy a nie suwaka (suwak jest dość problematyczny przy dłuższych nagraniach).



Rysunek 3.7. Pierwsze sekundy tego materiału: https://www.youtube.com/watch?v=MdLfWFzuGO8t=14s