SPIS TREŚCI

1. Wstęp

Celem pracy jest zaprojektowanie i implementacja adaptacyjnego systemu korekcji parametrów nagrań dźwiękowych. Algorytm dokonuje analizy akustyki pomieszczenia jako próbki nagrania audio w celu ekstrakcji parametrów akustycznych charakterystycznych dla tego nagrania, takich jak wpływ pomieszczenia na barwę dźwięku, czas pogłosu. Parametry te nadają nagraniu charakterystyczne brzmienie, które może zostać nałożone na czyste nagranie studyjne.

Projektowany system dokonuje korekty nagrania, rozumianej jako manipulacja badanych wcześniej parametrów dopasowując je według danych założeń do wzorca. System taki znajdzie zastosowanie między innymi jako wtyczka/rozszerzenie do programów obróbki dźwięku i będzie narzędziem efektowym wykorzystywanym do realizacji utworów muzycznych. Narzędzie to byłoby w stanie zarówno nakładać efekty pogłosowe dopasowane do pożądanego pomieszczenia, jak i również usuwać niechciany wpływ pomieszczenia na nagranie – usuwanie pogłosu z nagrania.

Nagranie poddawane analizie w początkowej fazie projektowania algorytmu jest odpowiedzią impulsową pomieszczenia, co dokładniej opisane zostanie w dalszej części pracy.

Analiza próbek nagrań dźwiękowych w następnych rozdziałach odbywa się w kontekście ludzkiego postrzegania zachodzących zjawisk, aniżeli ich opisu w zakresie fizyki. Dobór parametrów charakteryzujących badany sygnał ma więc na celu odzwierciedlenie elementów, które dobrze reprezentują postrzeganie zmiany zawartości sygnału po jego cyfrowej obróbce, a także zmiany sygnału dźwiękowego w zależności od cech pomieszczenia.

Eksperymentalnym elementem, o który można by rozszerzyć działanie algorytmu jest uczenie maszynowe, które nadal jest często poruszanym tematem i ze względu na badawczy charakter pracy oraz podejście różniące się od zgodnych ze sztuką i normami pomiarami mogłoby stanowić interesujący wpływ na wynik działania programu. Wytrenowanie modeli sieci neuronowych na podstawie subiektywnych ocen ankietowanych osób mogłoby usprawnić działanie algorytmu w kwestii odzwierciedlenia zmian subiektywnych cech dźwięku.

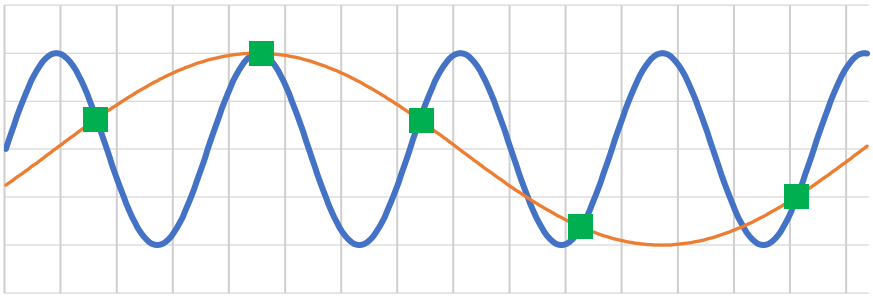
2. Teoria

Cyfrowe przetwarzanie sygnałów jest narzędziem, umożliwiającym dokonywanie złożonych analiz sygnałów, które w technice analogowej stanowiłyby trudność, albo byłyby wręcz niemożliwe do realizacji. W połączeniu ze współczesnymi wysokopoziomowymi językami programowania stanowią środowisko, którego jedynym ograniczeniem jest kreatywność programisty.

2.1. Definicje

Próbkowanie – dyskretyzacja sygnału w dziedzinach czasu i częstotliwości. Jest to niezbędny zabieg w technice cyfrowej, aby uzyskać skończoną liczbę danych umożliwiającą ich przetworzenie. Niewłaściwie dobrana częstotliwość próbkowania powoduje zjawisko aliasingu i w skrajnych przypadkach doprowadza do sytuacji, w której spróbkowany sygnał nie nadaje się do dalszego przetwarzania, gdyż nie odzwierciedla oryginału w taki sposób, w jaki oczekiwaliśmy.

Aliasing – aby sygnał cyfrowy został poprawnie odtworzony z postaci cyfrowej, najwyższa częstotliwość składowej sygnału musi równać się połowie częstotliwości próbkowania tegoż sygnału (częstotliwość Nyquista). W przeciwnym wypadku występuje zjawisko aliasingu, czyli błędnego odtworzenia spróbkowanego sygnału.



Wykres 1 Dwa sygnały sinusoidalne (niebieski, pomarańczowy) oraz próbki (zielony).

Na powyższym wykresie próbki przedstawione zielonym kolorem nie pozwalają na odtworzenie niebiskiej sinusoidy, gdyż w procesie rekonstrukcji sygnału z tychże próbek otrzymamy inną sinusoidę.

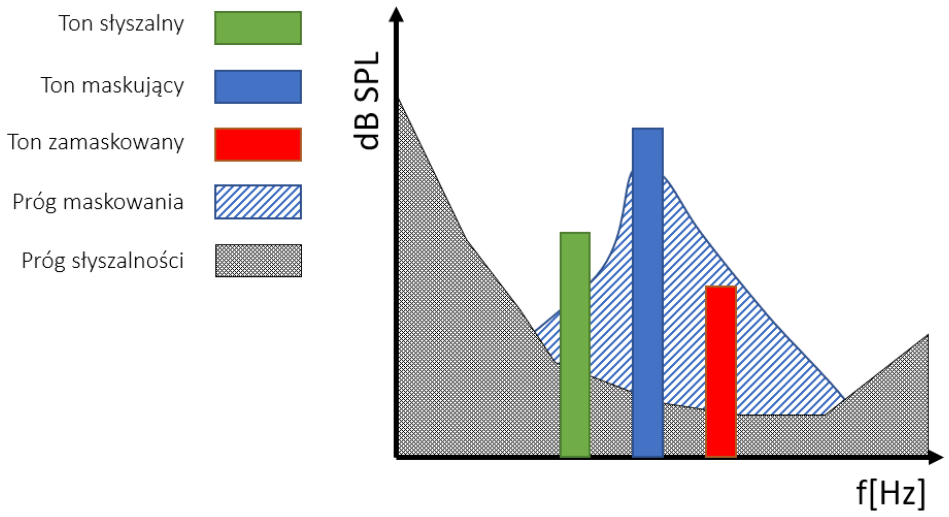
FFT – Fast Fourier Transform – Jest to algorytm służący do rozkładu sygnału na pojedyncze składowe częstotliwościowe. Jest to operacja matematyczna pozwalająca na przejście z dziedziny czasu na dziedzinę częstotliwości.

Dokonując operacji szybkiej transformaty Fouriera ważnym czynnikiem wpływającym na dokładność wyznaczonego widma sygnału jest ilość próbek w oknie czasowym FFT.

Kompresja dźwięku – dzieli się na dwie kategorie - kompresja stratna i bezstratna. Polega na zmniejszeniu zawartości redundantnych danych, bądź też usunięciu zbędnych danych w przypadku kompresji stratnej.

Format WAV - umożliwia zapis strumienia nieprzetworzonych danych. Zawiera on sygnały o wyższych częstotliwościach niż sygnał zakodowany w formacie MP3, a także zawiera dźwięki które są niesłyszane przez ludzkie ucho, takie jak tony zamaskowane tonem o wyższej amplitudzie, które kodowanie MP3 usuwa.

Maskowanie –



Wykres 2 Maskowanie tonem.

Czas pogłosu - definiuje się jako zanik energii o 60dB. Jest to tysiąckrotny zanik poziomu ciśnienia akustycznego, co zapisuje się:

$$10^{\frac{60\,dB}{20}} = 1000.$$

Parametr T30 - jest to czas, po jakim poziom energii spadnie o 60dB względem stanu początkowego, wyznaczany na podstawie nachylenia krzywej zaniku poziomu energii w

zakresie 30dB, co umożliwia wyznaczenie czasu pogłosu nawet w przypadku, gdy mamy do czynienia z szumami.

MLS – (Maximum Length Sequence) jest sygnałem losowym (bądź też pseudolosowym) którego funkcja autokorelacji jest równa 1 dla zerowego przesunięcia w czasie oraz bliska zeru dla pozostałych przesunięć w czasie, z tego powodu sygnał ten po operacji korelacji wzajemnej (ang. Cross-correlation) daje odpowiedź impulsową pomieszczenia.

Splot – operacja matematyczna określona wzorem:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t-\tau) \cdot g(\tau)\, d\tau,$$

która dla sygnału dyskretnego w czasie określa się jako:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m].$$

Przy implementacji algorytmów związanych ze splotem należy pamiętać o tym, że po przejściu na dziedzinę częstotliwości operacja splotu jest równoważna mnożeniu:

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\},$$

co upraszcza złożoność obliczeniową w aplikacji, która także wylicza FFT do innych celów.

Rozplot – dekonwolucja jest operacją odwrotną do funkcji splotu. W kontekście nagrań dźwiękowych można ją rozumieć jako usuwanie wprowadzonych zniekształceń sygnału w procesie splotu.

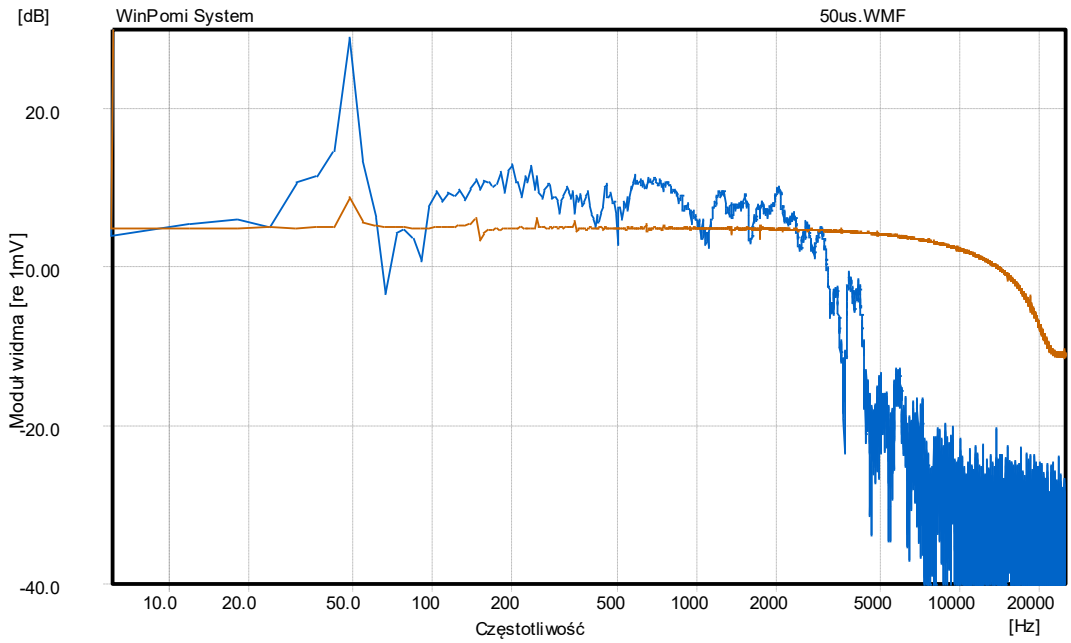
2.2. Metody pomiarowe odpowiedzi impulsowej pomieszczenia

Do pomiaru odpowiedzi impulsowej możemy użyć jednej z dwóch metod. Pierwszą z nich jest metoda szumu przerywanego, w której generujemy sygnał losowy zasilając głośnik wszechkierunkowy sygnałem losowym. Pomiary dokonywane są albo kolejno w pasmach oktawowych, bądź 1/3 oktawy, albo równocześnie we wszystkich pasmach przy użyciu szumu szerokopasmowego. Norma określa dokładnie, jak powinien wyglądać sygnał pobudzający oraz jaki powinien być czas pobudzenia. Drugą metodą jest całkowanie odpowiedzi impulsowej, gdzie sygnałami pomiarowymi mogą być: strzał z pistoletu, impuls szumu, sygnał chirp, czy też MLS.

2.3. Charakterystyka częstotliwościowa w zależności od długości impulsu

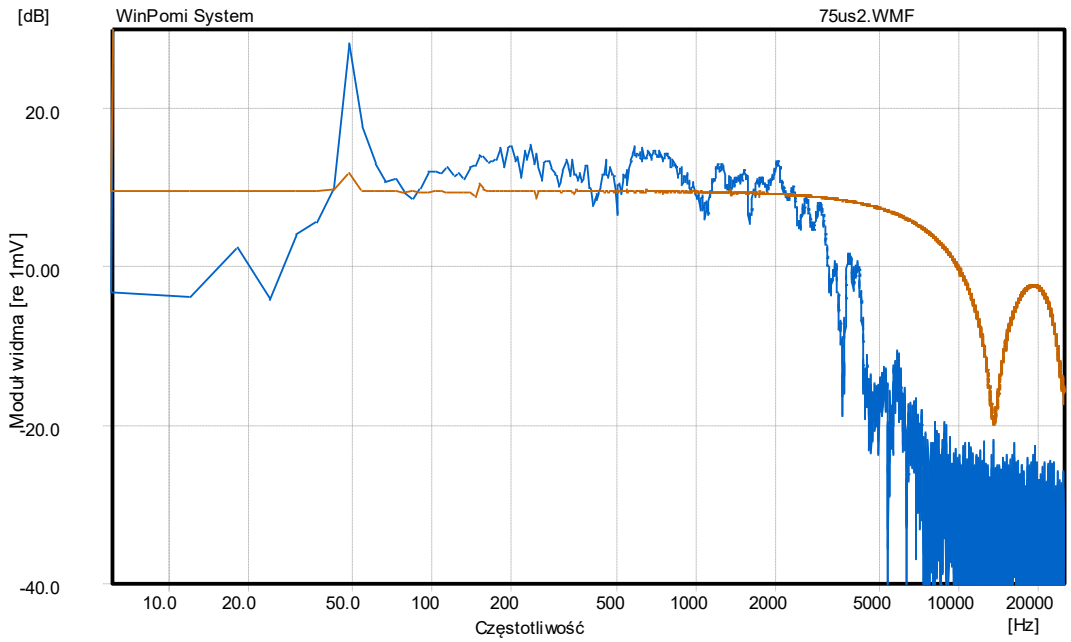
Do pomiaru czasu pogłosu metodą impulsową należy uwzględnić rodzaj źródła tegoż sygnału. W poniższych przykładach zaprezentowano sygnał impulsowy z generatora odtwarzany na głośnikach. Innym, podobnym źródłem sygnału impulsowego mogą także być: pękający balon czy wystrzał z pistoletu.

W zależności od charakteru źródła, możemy mieć do czynienia z bardzo wąskim pasmem użytecznych częstotliwości, bądź też niskim poziomem sygnału, gdy sygnał ma bardzo krótki czas trwania (co zaprezentowano na poniższych przykładach):

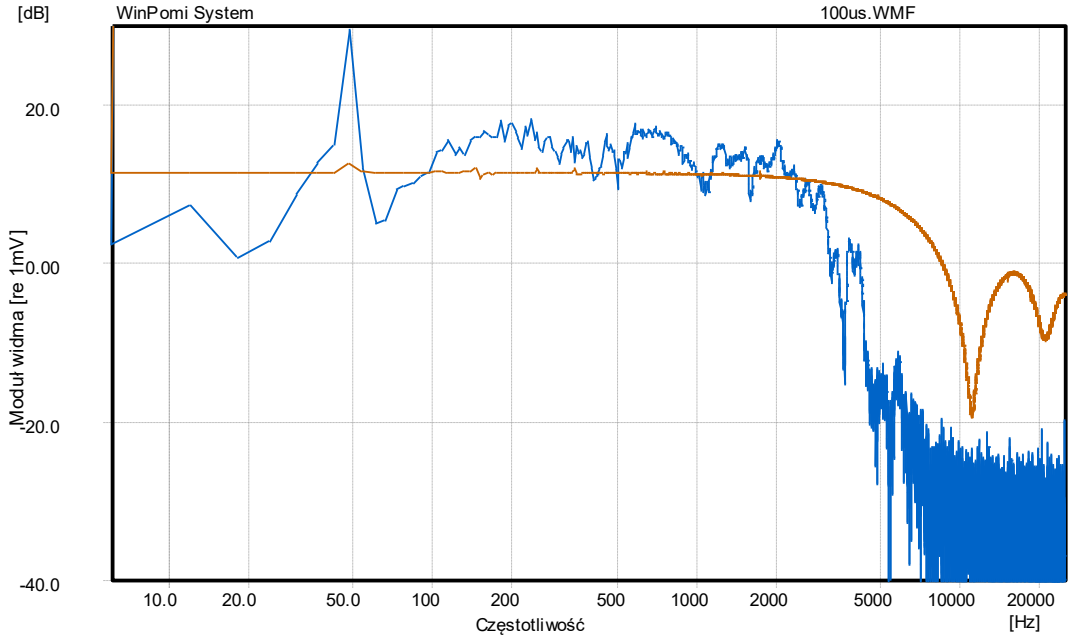


- *Rysunek 1 Charakterystyka częstotliwościowa dla impulsu 50 µs*

Na powyższym wykresie widać, że płaska, użyteczna część impulsu (sygnał wejściowy [pomarańczowy]) kończy się na około 10kHz. Na kolejnych wykresach zwiększa się czas trwania impulsu.

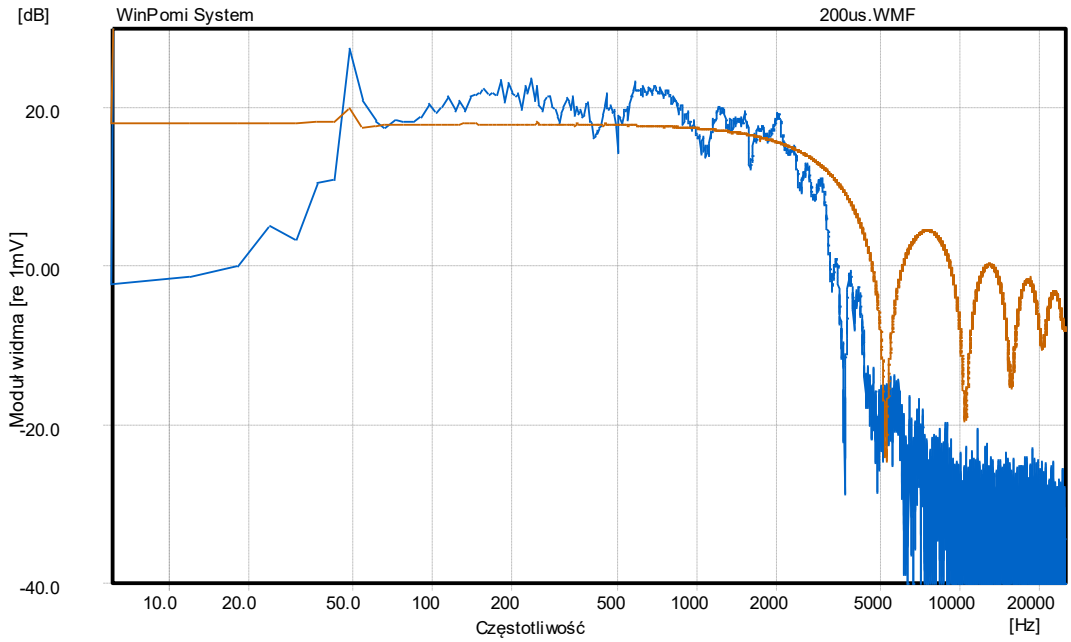


Rysunek 2 Charakterystyka częstotliwościowa dla impulsu 75 µs

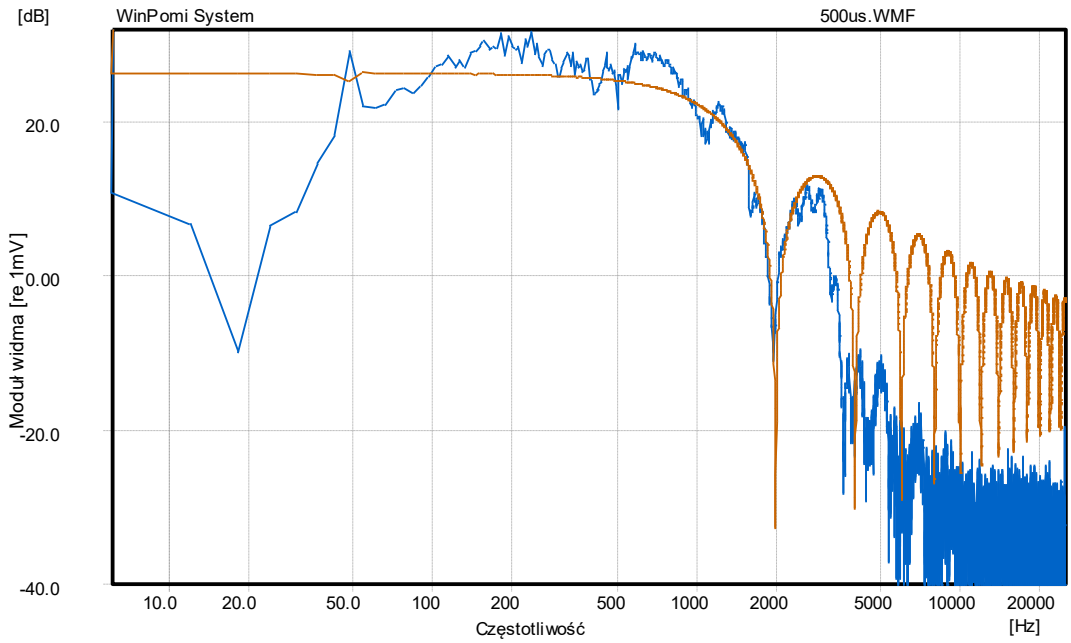


Rysunek 3 Charakterystyka częstotliwościowa dla impulsu 100 µs

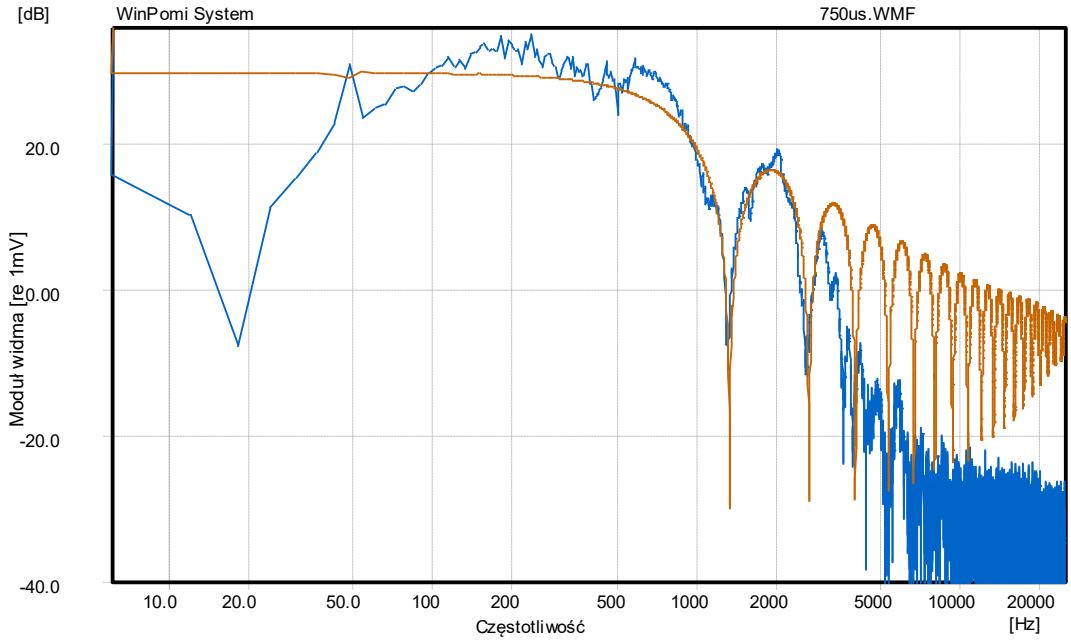
Wraz z wzrostem czasu trwania impulsu sygnału zaobserwować możemy skrócenie zakresu użytecznych częstotliwości, wynikiem czego sygnał będzie mniej użyteczny do pomiaru odpowiedzi impulsowej.



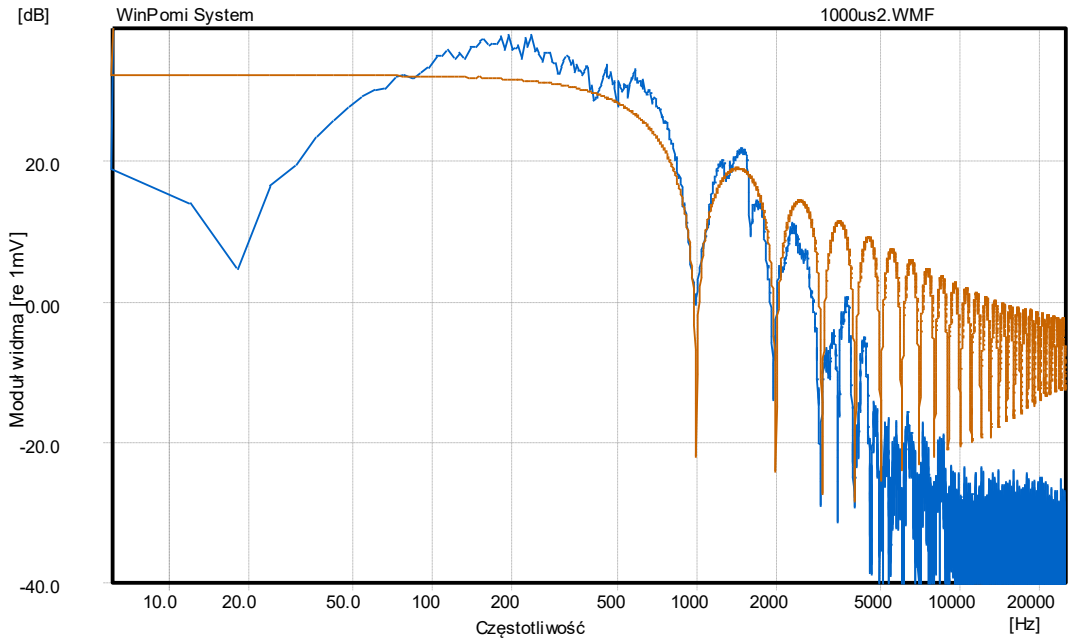
Rysunek 4 Charakterystyka częstotliwościowa dla impulsu 200 µs



Rysunek 5 Charakterystyka częstotliwościowa dla impulsu 500 µs



Rysunek 6 Charakterystyka częstotliwościowa dla impulsu 750 µs



Rysunek 7 Charakterystyka częstotliwościowa dla impulsu 1000 µs

Na podstawie zaprezentowanych powyżej pomiarów widać, iż ze wzrostem długości trwania impulsu pomiarowego skraca się użyteczny zakres pasma częstotliwości mających płaską charakterystykę, co stanowi problem, gdy chcemy użyć takiego sygnału do pomiaru odpowiedzi impulsowej pomieszczenia. Niemniej jednak sygnał ten został wybrany, gdyż odpowiednio dobrany spełnia założenia przedstawione w normie EN ISO 3382, która stwierdza, iż wystarczy źródło impulsowe potrafiące wytworzyć sygnał, taki, aby krzywa zaniku poziomu energii zaczynała się co najmniej 45dB powyżej tła akustycznego pomieszczenia.

2.4. Barwa dźwięku

Barwa dźwięku jest subiektywną cechą, którą każda osoba może postrzegać w znacznie różniący się od siebie sposób, stąd też ciężko zdefiniować jest dokładną wartość togo parametru.

Instrumenty muzyczne potrafią wytworzyć ton o tej samej wysokości dźwięku, niemniej jednak dźwięki te będzie rozróżniać właśnie barwa, na którą wpływ ma między innymi szerokość widma emitowanego sygnału, a także zawartość i amplituda składowych harmonicznych w tymże sygnale.

2.4.a. Spectral centroid

Spectral centroid, czyli środek ciężkości charakterystyki częstotliwościowej sygnału jest parametrem, który został użyty w algorytmie w celu aproksymacji barwy dźwięku rozumianej jako częstotliwości dominujące w danym nagraniu w określonym odcinku czasu.

$$\frac{\sum_{n=0}^{N-1} f(n) \cdot x(n)}{\sum_{n=0}^{N-1} x\,(n)}$$

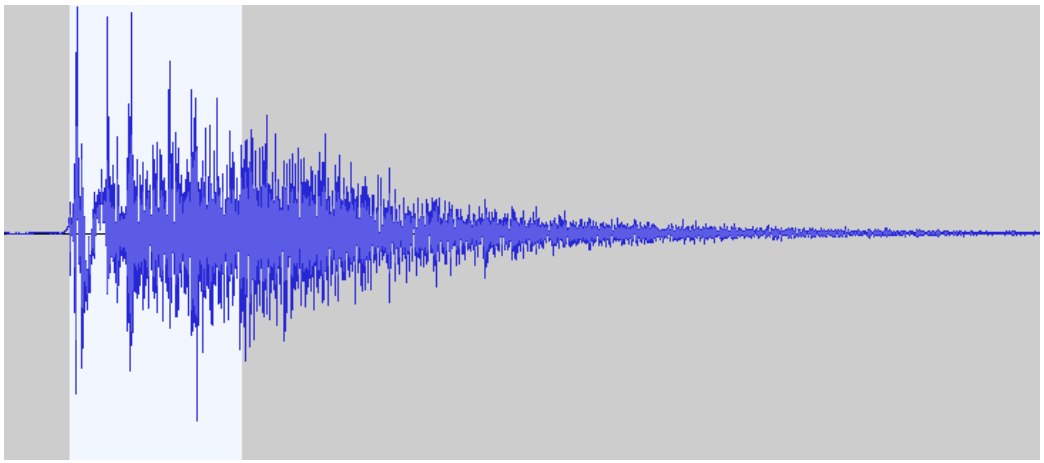
Algorytm został zmodyfikowany, aby lepiej odzwierciedlał wrażenie wysokości dźwięku percepowanego przez słuchacza, co zostanie opisane w

3. Projekt aplikacji

Do implementacji projektu wybrano język Python w wersji *3,7*, który charakteryzuje się szerokim zakresem zastosowań ze względu na wysokopoziomową składnię, pozwalającą na szybką implementację złożonych czy rozbudowanych aplikacji. Wsparcie w postaci różnorodnych i specjalistycznych bibliotek dla tego języka umożliwia znacząco szybsze rozwiązywanie zaawansowanych problemów, które na przykład w przypadku języka *C++* wymagałyby wielokrotnie więcej nakładu pracy (co mogłoby być niewspółmierne do zalet płynących z zastosowania *C++*). Wersja *3,7* zapewnia wsparcie w możliwym późniejszym rozwoju aplikacji.

3.1. Możliwości

Zrozumiałość mowy w danym pomieszczeniu najbardziej zależna jest od powstających wczesnych odbić, które definiują stosunek oryginalnego sygnału do jego ogona pogłosowego. Analizując odpowiedź impulsową pomieszczenia możemy więc analizować sygnał pod tym kątem.

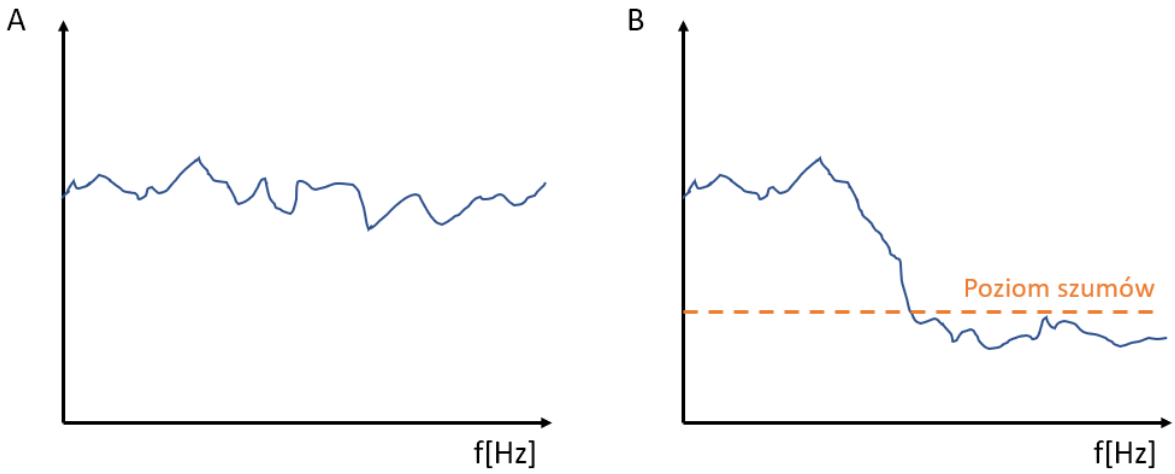


Rysunek 8 Nagranie źródła impulsowego w pomieszczeniu.

Dokładniejszą metodą analizy powstających odbić jest zastosowanie sygnału sinusoidalnego w postaci krótkiego impulsu, który można następnie przeanalizować dokonując korelacji z czystą reprezentacją tego sygnału.

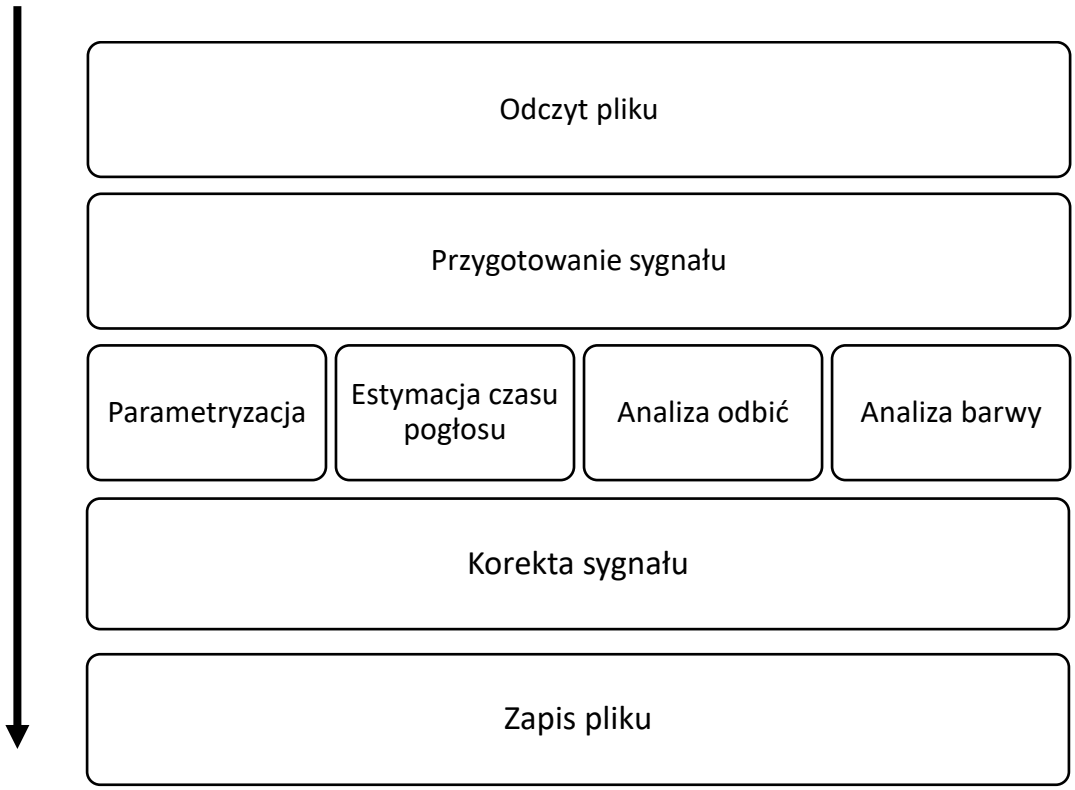
Ogromne wyzwanie stanowi problem, który znacznie utrudnia dokonania rozplotu nawet, gdy znamy odpowiedź impulsową pomieszczania. Tym problemem jest obecność szumu w nagraniu.

W przypadku, gdy operacja splotu będzie miała charakter filtru dolnoprzepustowego w operacji odwrotnej musimy podbić wyższe częstotliwości, aby odtworzyć oryginalny sygnał. Jednakże, jeżeli amplituda sygnału w tym paśmie jest na poziomie szumów sygnału, albo i nawet poniżej jego poziomu, to sygnał został bezpowrotnie utracony. W próbie podbicia tych częstotliwości wzmocnimy jedynie poziom szumów.



Taki efekt został uzyskany przy próbie usunięcia pogłosu z nagrania metodą rozplotu z odpowiedzią impulsową pomieszczenia. Filtrowanie szumów z nagrania nie przyniosło większej różnicy, efekt wzmocnienia szumów zmienił się w rozbrzmiewanie dzwonienia.

3.2. Schemat działania



Działanie algorytmu w projekcie aplikacji zaplanowano na podejście do warstwowej implementacji kodu. Pierwsza warstwa bezpośrednio odpowiada za odczyt pliku z nagraniem (z założenia w formacie wave) i zaimplementowana została jako klasa ***WaveFile*** odczytująca i przechowująca parametry zapisu pliku jako zmienne obiektu klasy, które będą mogły zostać wykorzystane przez następne warstwy programu (dokładny opis szczegółów implementacji zostanie rozwinięty w dalszej części pracy). Kolejną warstwę stanowi między innymi klasa Signal, która odpowiada za przygotowanie danych dotyczących sygnału zawartego w obiekcie klasy WaveFile (między innymi posiada metodę ***timeLine()*** przygotowującą sygnał do wyświetlenia na wykresie).

3.3. Implementacja

W pierwszym kroku algorytm dokonuje odczytu pliku w formacie *wave*, który umożliwia zapis nagrania bez kompresji.

Do wykonania tego etapu utworzono klasę WaveFile, która inicjowana jest adresem do pliku:

```
class WaveFile:
    PATH = str()

    def __init__(self, path):
        self.PATH = path
        self.FILE = wave.open(self.PATH)
        [self.CHANNELS, self.SAMP_WIDTH, self.FRAMERATE, self.NFRAMES,
*self.rest] = self.FILE.getparams()
        self.RAW = np.fromstring(self.FILE.readframes(-1), dtype='int32')
        self.RAW = self.RAW/max(self.RAW)
```

Instancja tejże klasy przechowuje informacje na temat sposobu w jakim zostało zapisane nagranie. Jest to między innymi częstotliwość próbkowania sygnału, która będzie niezbędnym parametrem w dalszych etapach działania programu.

Przykład utworzenia instancji klasy:

```
orig_file = WaveFile('./PathToFile/file.wav')
```

Opis wykorzystanych zmiennych:

- PATH - ścieżka do pliku,
- FILE - obiekt modułu wave,
- CHANNELS - liczba kanałów,
- SAMP_WIDTH - ilość bajtów dla jednej próbki,
- FRAMERATE - częstotliwość próbkowania,
- NFRAMES - liczba próbek,
- RAW - nieprzetworzony zapis pliku wave.

Odczyt wartości próbek zapisany do zmiennej *RAW* (będącej jednowymiarową tabelą typu *int32*[1]), która w następnym kroku algorytmu zostaje znormalizowana (w wyniku operacji dzielenia typ zmiennej ulega rzutowaniu na typ *float*).

Celem utworzenia tejże klasy było określenie dolnej warstwy działania programu, która pozwala na wczytane pliku wraz z załadowaniem jego niezbędnych parametrów, które ułatwią utworzenie kolejnej warstwy działania algorytmu dokonującej już analiz samego sygnału.

Klasa **Signal** stanowi kolejną warstwę algorytmu. Przy inicjacji nowego obiektu tej klasy jako argument zadajemy obiekt kasy WaveFile (**file**: WaveFile), podpis/tytuł sygnału (title: **str**) oraz boolean definiujący czy oś x będzie osią czasu (timeDomain: **bool**).

```
class Signal:
    def __init__(self, file: WaveFile, title: str, timeDomain: bool):
        self.title = title
        self.timedomain = timeDomain
        self.data = file.RAW
        self.fp = file.FRAMERATE

        def timeLine(Y, fp):
            return np.linspace(0, len(Y) / fp, num=len(Y))

        if timeDomain:
            self.time = self.timeLine(self.data, self.fp)
        else:
            self.time = -1
```

Przykład utworzenia obiektu klasy:

```
sig_1 = Signal(file=orig_file,
               title="Original file",
               timeDomain=True)
```

, gdzie zadano plik wejściowy orig_file mający reprezentację w dziedzinie czasu.

[1] 32 bitowa liczba całkowita z zakresu od -2^{31} do $2^{31}-1$.

Do wyliczenia całki w algorytmie opisanym w *3.3.a Pogłos - Badanie odpowiedzi impulsowej* utworzono funkcję:

```
def integrate(f):
    integral = np.zeros_like(f)

    ox = range(len(f))

    print("wait - calculating integral - {} samples".format(len(f)))

    for i in range(1, len(f)):
        y = simps(f[0:i], ox[0:i])
        integral[len(f)-i] = y
    print("done")
    #db
    integral_max=max(integral)
    for i, val in enumerate(integral):
        if val > 1e-10:
            s = val / integral_max
            l = math.log10(s)
            integral[i] = 10 * l
        else:
            integral[i] = -80
    return integral
```

Użyte funkcje z bibliotek:

Funkcja FFT zwracająca jednowymiarową tablicę rzeczywistej części wyniku działania:

```
def rfft(a, n=None, axis=-1, norm=None):
```

a – tablica, *n* – długość FFT, *axis* - oś, według której wyliczane jest FFT, *norm* - normalizacja
Całkowanie metodą Simpsona (przybliżanie wartości całki oznaczonej funkcją kwadratową):

```
def simps(y, x=None, dx=1, axis=-1, even='avg'):
```

y – tablica, *x* – punkty próbkowania *y*, even – metoda uśredniania

3.3.a. Pogłos - Badanie odpowiedzi impulsowej

Algorytm wyliczania czas pogłosu wzorowany jest na opisie tego procesu zawartego w normie *PN-EN ISO 3382*. W pierwszym kroku wartości wszystkich próbek podnoszone są do kwadratu celem usunięcia wartości ujemnych.

```
for i, val in enumerate(frames):
    f_squared[i] = do_square(val)
```

Następnie odwrócono kolejność próbek, gdyż chcemy dokonać operacji całkowania odwrotnej w czasie uzyskując w ten sposób zanik energii.

```
f_squared = np.flip(f_squared)
```

Po czym dokonano całkowania i utworzono oś czasu X:

```
integral = integrate(f_squared)
time = np.linspace(0, len(integral)/FP, num=len(integral))
```

```
db30o = x[y.searchsorted(-30, 'left')]
db30 = round(db30o*1000*CHOP)
db0o = x[y.searchsorted(-0.2, 'right')]
db0 = round(db0o*1000*CHOP)
print("0dB---> at: " + str(db0) + "ms")
print("-30dB-> at: " + str(db30) + "ms")
print("-60dB-> at: " + str(db30+(db30-db0)) + "ms [extrapolated]")
print("T30------>: " + str((db30-db0)*2) + "ms")

x1 = int((db30o)*FP)                    # spadek  o 30dB
x2 = int((db0o)*FP)                     # odniesienie
x3 = int((db0o+((db30o-db0o)*2))*FP)    # spadek o 60dB szacowany
```

Na pliku reprezentującym odpowiedź impulsową pomieszczenia przeprowadzana zostaje operacja całkowania.

$$E(t) = \int_{t+T_0}^{t} p^2(\tau)\, d(-\tau) \qquad (1)$$

Równanie 1 PN-EN ISO 3382

Z wykresu krzywej zaniku poziomu energii szacowany jest spadek o 60dB na podstawie zakresu 30dB [parametr T30].

3.3.a. Przykładowy pomiar zaniku poziomu energii w pomieszczeniu.

W celu uzyskania odniesienia, co do dokładności oraz poprawności działania algorytmu określania czasu pogłosu przeprowadzono pomiary przy użyciu sprzętu laboratoryjnego wraz z profesjonalnym oprogramowaniem.

Poniżej przedstawiono parametry pomieszczenia, w którym wykonano pomiary pogłosu zarówno sprzętem profesjonalnym, jak i za pomocą opisywanego w pracy systemu. Wyniki oraz różnice w pomiarach zostaną przedstawione w dalszych częściach pracy.

Tabela 1 Pomiar wymiarów pomieszczenia.

wymiary pomieszczenia [m]	
długość	5,92
szerokość	5,40
wysokość	3,40

Na podstawie danych z tabel można by już zgodnie z teorią statystyczną skorzystać ze wzoru Sabina na czas pogłosu:

$$T = \frac{0{,}161V}{S\bar{\alpha}}[s], \qquad \bar{\alpha} = \frac{1}{S}\sum_{i=1}^{n}\alpha_i S_i$$

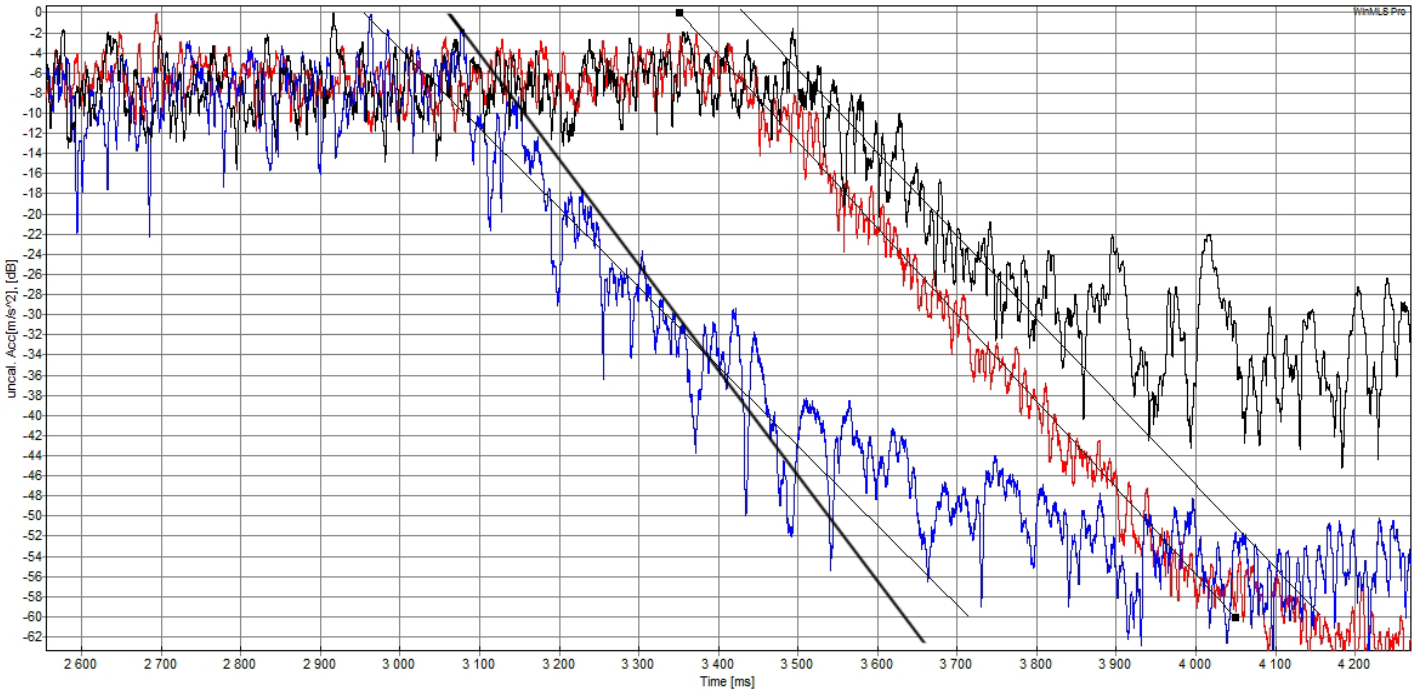
T - czas pogłosu, V- objętość pomieszczenia, S – powierzchnia, $\bar{\alpha}$ – średni współczynnik pochłaniania, α – chłonność akustyczna

Tabela 2 Współczynniki pochłaniania wybranych elementów pomieszczenia.

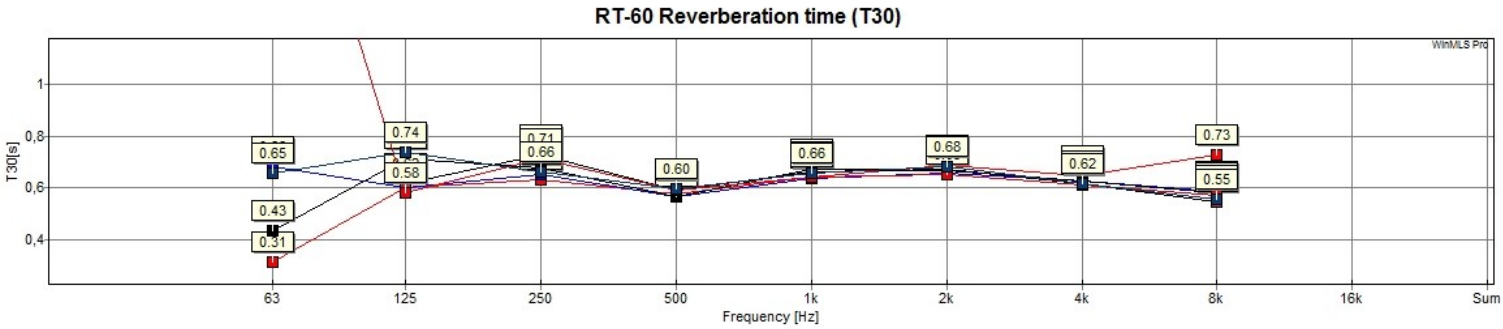
	wsp. pochłaniania	**powierzchnia [m^2]**
drzwi	0,18	2,03
meble	0,17	7,00
okno	0,18	7,16
ściany	0,02	59,04
drzwi	0,17	1,76
podłoga	0,31	31,97
sufit	0,05	31,97

Poniżej zaprezentowano wzorcowy pomiar czasu pogłosu metodą impulsową wykonany na profesjonalnym sprzęcie, zgodnie z normą *PN-EN ISO 3382*.

Nagrania wykonano w trzech miejscach położenia mikrofonu oraz trzech miejscach położenia źródła dźwięku (co daje razem 9 nagrań na jedno pomieszczenie).



Wykres 4. Krzywe zaniku poziomu energii [czerwona - 2kHz, niebieska - 500Hz, czarna – bez filtra]



Wykres 3 Parametr T60 w pasmach częstotliwości.

Uśrednione wartości wykonanych pomiarów z zakresów częstotliwości:

Tabela 3 Uśrednione wartości parametru T30

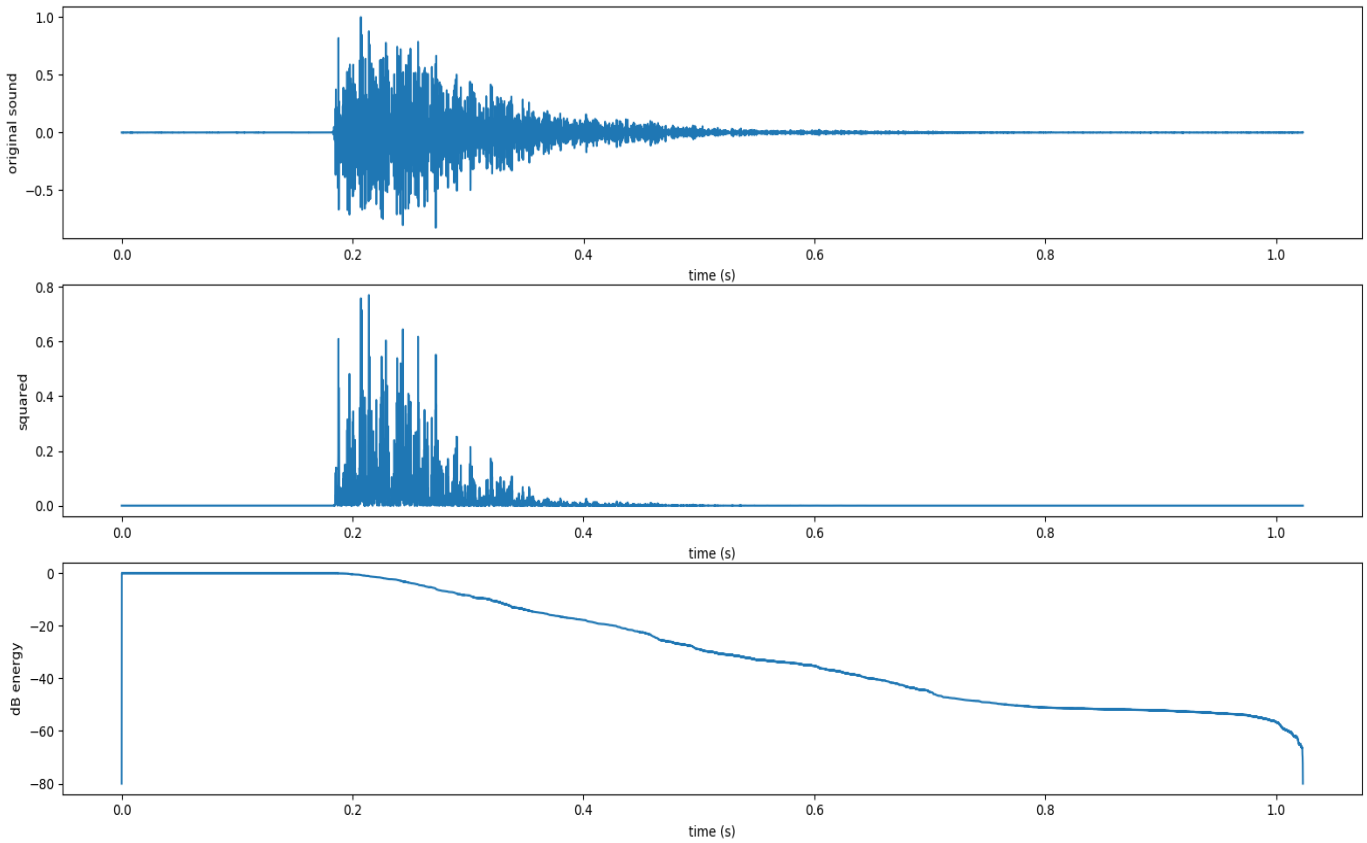
Parametr	częstotliwość [Hz]							
	63	125	250	500	1000	2000	4000	8000
T30[s]	0,72	0,64	0,68	0,58	0,65	0,67	0,63	0,59

3.3.b. Pomiar pogłosu poprzez omawiany program

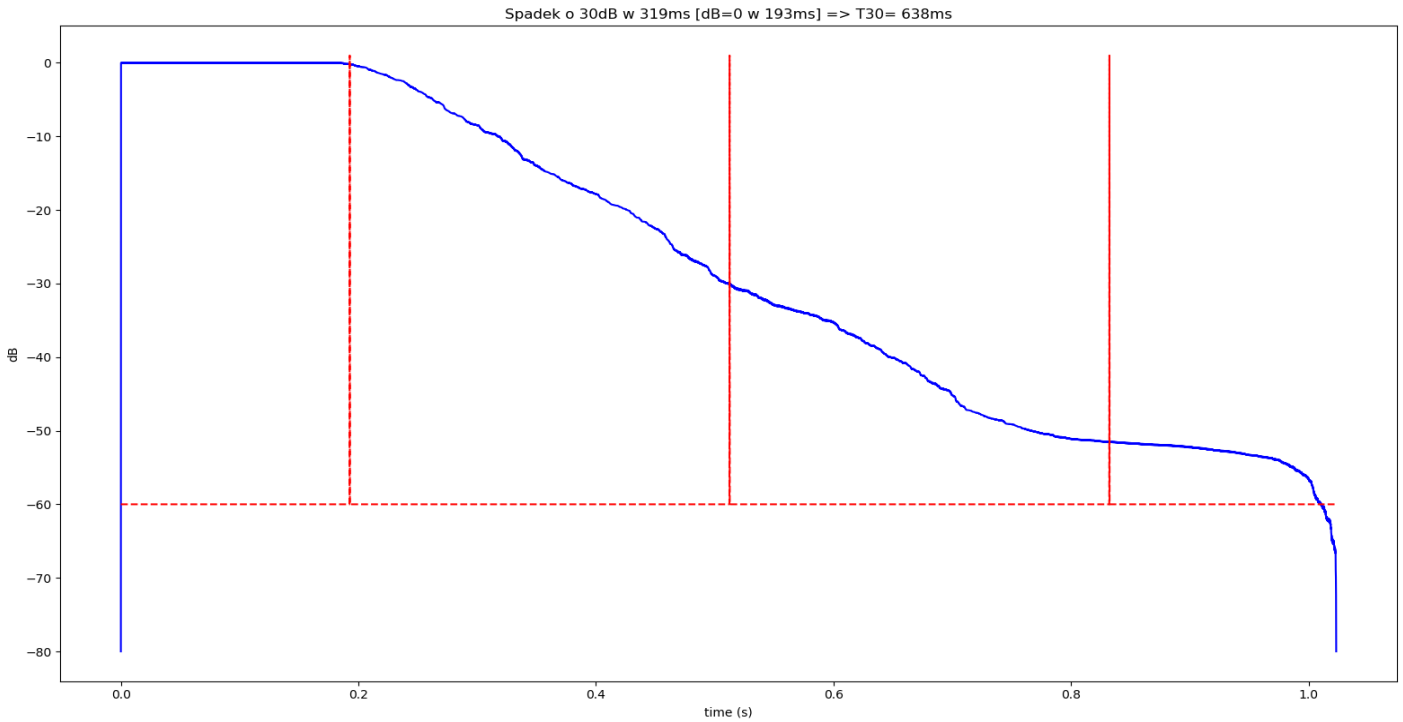
Algorytm wyliczający czas pogłosu bazuje na metodzie impulsowej, która pozwala na analizę wyników w wielu pasmach częstotliwości, które dla pomiarów metodą szumu przerywanego trzeba wykonywać osobno.

Do wykonania nagrań odpowiedzi impulsowej użyto źródła dźwięku różnego rodzaju, między innymi – pękający balon, trzaśnięcie, klaśnięcie, aby oszacować wpływ jakości nagrania odpowiedzi impulsowej na późniejszą operację splotu [całe badanie ma charakter subiektywnego postrzegania działania algorytmu, aniżeli odwzorowywania pomiarów zgodnie ze sztuką].

Do zminimalizowania poziomu tła w nagraniach dokonano analizy w pasmach częstotliwościowych przy użyciu filtrów cyfrowych, co pozwoliło na uzyskanie większego zakresu (około 30-50dB) zaniku poziomu dźwięku pozwalającego oszacować jego spadek o 60dB.



Wykres 5 Reprezentacja graficzna kolejnych etapów obliczania zaniku poziomu dźwięku. Kolejno: dane wejściowe, podniesienie do kwadratu, całka odwrotna w czasie.



Wykres 6 Wynik algorytmu obliczającego parametr T30.

Algorytm określił kolejne wartości:

Poziom [dB]	Czas[ms]
0	193,0
-30	512,0
-60 [T30]	638,0

Gdzie poziom w decybelach jest mierzony względem najwyższej odnotowanej amplitudy sygnału

$$L = 10\,log_{10}\left(\frac{I}{I_0}\right), I_0 = \mathrm{V}_{max}$$

Z założeń projektowych algorytmu, przy analizie nagrania pominięto częstotliwości mniejsze, jak 120Hz ze względu na dodatkowe trudności występujące w analizie tychże częstotliwości, takie jak rezonanse osiowe, styczne i skośne, a także wpływ jakości mikrofonu na pomiar częstotliwości w tym zakresie. Zjawiska te mogą wpływać na dodatkowe błędy, które utrudnią analizę działania projektowanego algorytmu, z tego powodu zostaną one pominięte, a analiza zostanie przeprowadzana w przypadkach „idealnych", czyli niestwarzających dodatkowych problemów.

3.3.c. Analiza zmiany barwy

Do algorytmu wyznaczającego parametr centroidy częstotliwościowej dodano ważenie amplitudy zgodnie z założeniami prawa Webera Fechnera, zapisanego wzorem:

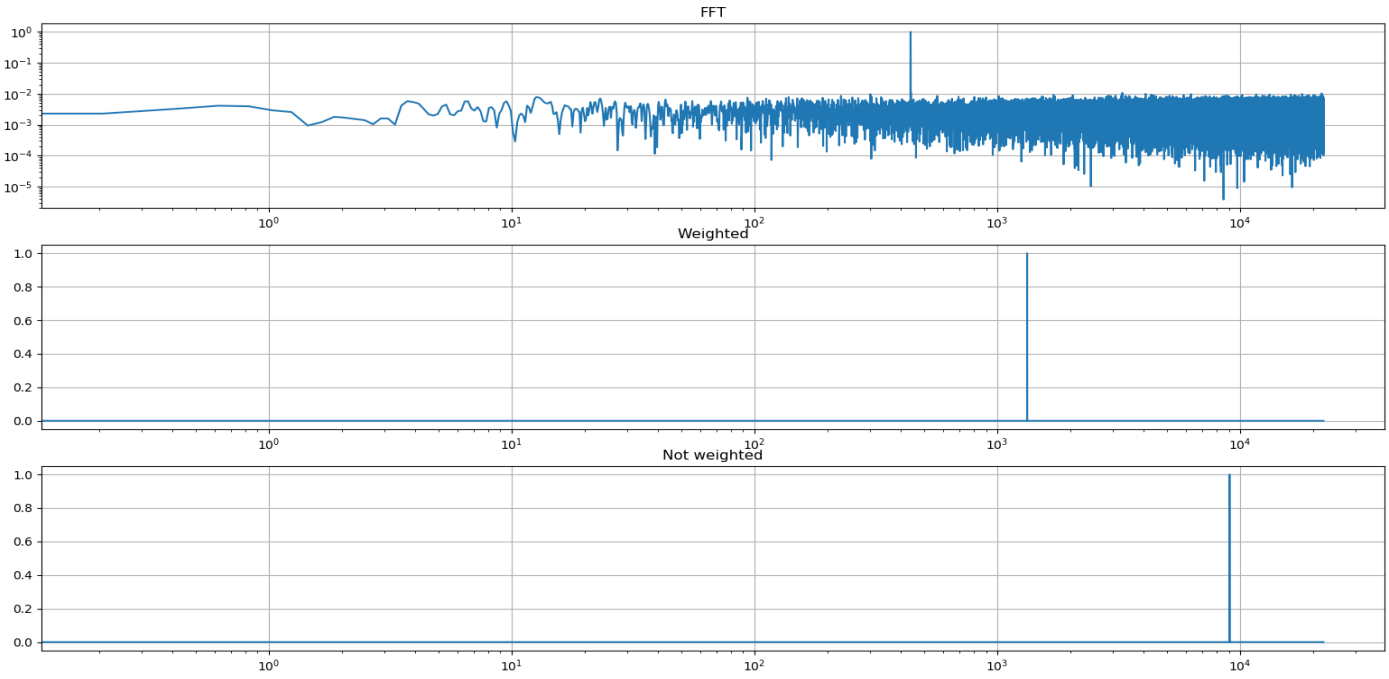
$$\omega = k \cdot ln\left(\frac{B}{B_0}\right)$$

, gdzie k jest współczynnikiem proporcjonalności (percepcji wrażenia do faktycznej zmiany bodźca), a B jest wielkością bodźca.

Ze względu na to, iż rozdzielczość *FFT* na wykresie wynosi *10Hz*, co każdy punkt. Stąd też występuje zagęszczenie pomiarów dla wyższych częstotliwości dla skali logarytmicznej (która reprezentuje sposób postrzegana wrażeń dźwiękowych przez człowieka). Stąd, aby uzyskać bardziej naturalny dla ludzkiego ucha wynik działania tego algorytmu dodano ważenie wartości amplitudy *FFT* w odniesieniu do częstotliwości.

Przykłady

Przykład na wykresie [sinus 440Hz + szum biały] (kolejno FFT sygnału, watość SC ważone, wartość SC nieważone):

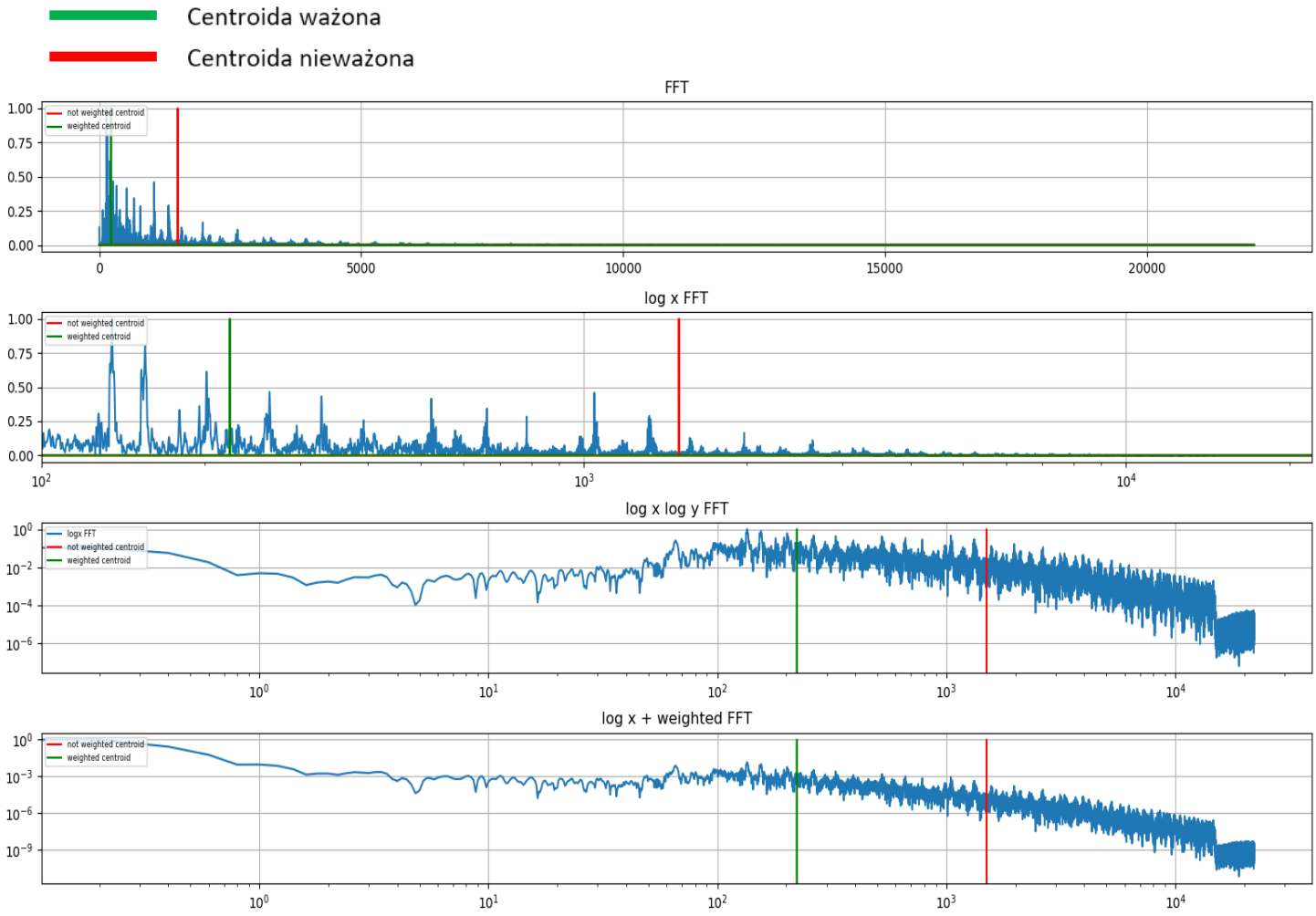


Wykres 7 Parametr Spectral Centroid obliczany z FFT.

Prawo Webera Frechnera odwołuje się do ludzkiej percepcji zagęszczenia danego bodźca. W kontekście percepcji częstotliwości jest to na przykład zmiana tonu z 200 Hz na 400Hz, która na pewno zostanie odnotowana przez słuchacza, natomiast zmiana z 8000 na 8200Hz już niekoniecznie, pomimo iż jest to nadal zmiana o 200Hz. Ważnym aspektem tego zjawiska jest względna zmiana intensywności bodźca.

Kolejny przykład – utwór muzyczny, kolejno:

- FFT (X lin Y lin) [wartości Y nieważone]
- FFT (X log Y lin) [wartości Y nieważone]
- FFT (X log i Y log) [wartości Y nieważone]
- FFT (X log i Y log) [wartości Y ważone]



Wykres 8 Działanie algorytmu obliczania parametru Spectral centroid.

Poprawka ta widocznie udoskonala znajdowanie środka wagi widma, tak aby reprezentowało ono postrzegane przez słuchacza dominujące składowe częstotliwościowe sygnału.

W obecności harmonicznych w sygnale, jak na przykład w tonie zagranym przez skrzypce oprócz wskazania częstotliwości o najwyższej amplitudzie zawartej w sygnale parametr ten uwzględni także zawartość harmonicznych, gdyż ich obecność przesunie środek wagi charakterystyki częstotliwościowej w stronę wyższych częstotliwości, co w rezultacie da różny wynik dla różnych instrumentów grających ten sam ton.

Poniżej przedstawiono kod implementacji.

```
def spectral_centroid1(f, y):

# nowa tablica
    New_y= np.zeros_like(y)

# Weber Fechner Law
    for i, magnitude in enumerate(y):
        waga = 0.01
        if i > 0:
            waga = (f[i]-f[i-1])/f[i]

# każdy y jest ważony według założenia prawa Webera Frechnera
        New_y[i] = y[i]*waga

    return np.sum((New_y * f) / np.sum(New_y)), New_y
```

3.3.d. Pogłos

Oddanie naturalności brzmienia, w kontekście zjawiska pogłosu wymaga między innymi rozpatrzenia jednego z ważnych elementów, jakim jest opóźnienie pomiędzy źródłem dźwięku, a pierwszym odbiciem, które to nadaje wrażenie wielkości pomieszczenia.

Fala dźwiękowa pokonując bezpośrednią drogę dociera najszybciej do słuchacza, następnie docierają wczesne odbicia.

Kolejnym elementem jest gęstość ech występujących w ogonie pogłosowym, która jest wymagana, aby dźwięk wydawał się naturalny i aby nie wystąpił efekt flutter. Zgodnie z badaniami Schroedera szacuje się, że gęstość ta wynosi około 1000 ech na sekundę (2).

3.3.e. Transformacja sygnału

Nałożenie zmierzonej odpowiedzi impulsowej na sygnał nagrany w warunkach zbliżonych do warunków studyjnych (niezauważalny pogłos oraz brak zniekształceń sygnału).

W celu przyspieszenia obliczeń (jako że mamy już wyliczone FFT we wcześniejszych etapach działania algorytmu) korzystamy z własności splotu:

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}, \text{ stąd } f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}.$$

W następnym kroku dokonujemy operacji odwrotnej transformacji Fouriera ponownie przechodząc na dziedzinę czasu, co w algorytmie zostało oznaczone:

$$\boldsymbol{y(t)} = h(t) * x(t) \equiv \boldsymbol{Y(s)} = H(s) \cdot X(s)$$

Poniżej przedstawiono fragmenty kodu implementacji.

```
def convolution(x, h):
    L = len(x) - 1  # określenie długości splotu
    N = nextpow2(L) # długość niech będzie potęgą dwójki

    H = np.fft.rfft(h, N)  # FFT nagrania impulsu
    X = np.fft.rfft(x, N)  # FFT sygnału wejściowego
    # Normalizacja
    H = H/max(H)
    X = X/max(X)
    for i, value in enumerate(H):
        if H[i] < 0.1:
            H[i]=1e-10

    Y = H * X  # operacja splotu
    y = np.fft.irfft(Y)  # powrót do dziedziny czasu

    y = np.array(y/(max(y)*1.001), dtype='float32')
    return y
```

Funkcja znajdująca potęgę liczby 2 najbliższą zadanej wartości L w celu określenia długości FFT.

```
def nextpow2(L):
    N = 2
    # Dopóki N mniejsze od L podnoś N do kwadratu
    while N < L: N *= 2
    return N
```

```
splot = convolution(orig_file.RAW, IR_inverted)

# przedstawienie wyniku działania na wykresie
plot(orig_file.RAW,IR_file.RAW,  splot)
```

Wynikiem działania tej części algorytmu jest plik dźwiękowy (zmodyfikowanego nagrania wejściowego) z nałożoną charakterystyką pomieszczenia poprzez splot z odpowiedzią impulsową danego pomieszczenia.

Rezultat jest zadowalający pod względem naturalności brzmienia pogłosu oraz charakterystyki pomieszczenia, takiej jak przygłuszenie wyższych częstotliwości dających wrażenie wielkości pomieszczenia.

Do łatwej analizy wyniku działania programu użyto biblioteki *matplotlib* dostarczającej łatwych narzędzi do tworzenia wykresów. Zaimplementowano funkcje *plot* oraz *plot_signals* umożliwiające tworzenie wykresów za pomocą wcześniej wspomnianej biblioteki bezpośrednio na instancjach klasy *Signal*.

Funkcja prezentując dowolną liczbę wykresów przy użyciu biblioteki **matpotlib**.

```
def plot(toPlot, *args):

    print(len(args))
    N_PLOTS = 1 + len(args)

    plt.subplot(N_PLOTS, 1, 1)
    plt.plot(timeLine(toPlot, 44100),toPlot)

    for index, pl in enumerate(args):

        plt.subplot(N_PLOTS, 1, index+2)
        plt.plot(pl)

    plt.show()
```

Funkcja tworząca wykresy z obiektów klasy **Signal** (która zawiera w sobie dane takie, jak: time, data, title (czas, dane, tytuł)).

```
def plot_signals(*args: Signal):

    # dla każdego Sygnału
    for i, arg in enumerate(args):
        plt.subplot(len(args),1, i+1)
        plt.title(arg.title)

         # podpisywanie wykresów
        if arg.timedomain: # jeżeli x jest osią czasu
            plt.xlabel('time [s]')
            plt.plot(arg.time, arg.data)
        else:              # jeżeli x nie jest osią czasu
            plt.plot(arg.data)
            plt.xlabel('frequency [Hz]')
    # wykreśl wykres
    plt.show()
```

```
# tworzenie obiektów klasy WaveFile (dodawanie plików)
orig_file = WaveFile(ORIG)
IR_file = WaveFile(IR)

# tworzenie obiektów klasy Signal (sygnałów) z klas WaveFile
sig_1 = Signal(file=orig_file,
               title="Original file",
               timeDomain=True)
sig_2 = Signal(file=IR_file,
               title="Impulse response",
               timeDomain=True)

plot_signals(sig_1, sig_2)
```

3.3.f. Zapis nowego pliku

```
wav_file = wave.open("file2.wav", 'w')

nchannels = 1
sampwidth = 2
framerate = 44100
nframes = len(splot)

wav_file.setparams((nchannels,
                    sampwidth,
                    framerate,
                    nframes,
                    comptype,
                    compname
                    ))
```

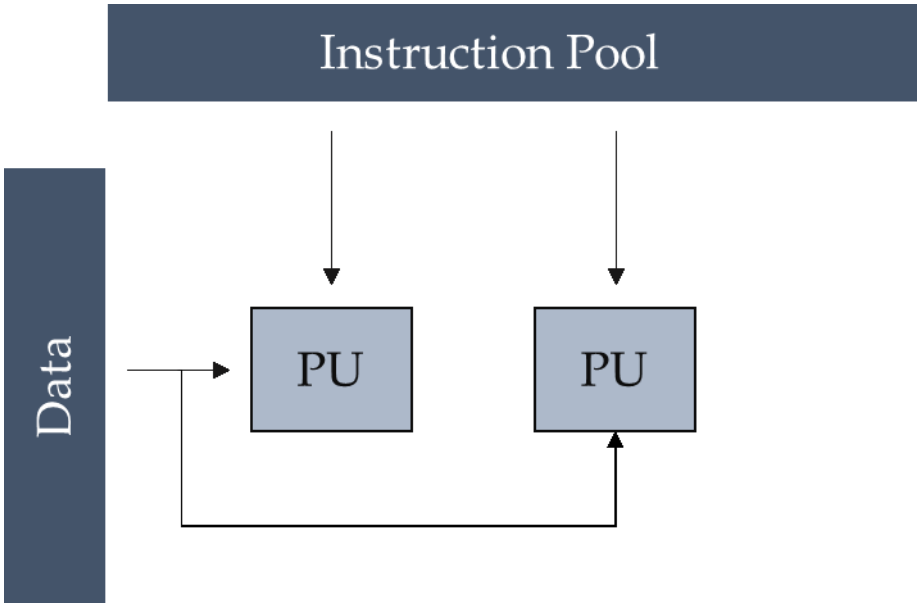
Zapisu dokonujemy z tymi samymi parametrami, co plik oryginalny (który to został użyty do zainicjowania obiektu klasy WaveFile)

```
wav_file.writeframes(np.array(splot, dtype='float32'))
wav_file.close()
```

3.4. Równoległość parametryzowania

W przypadku analizy plików o większej objętości czas obliczeń można skrócić stosując zrównoleglenie procesu parametryzacji. Obecnie równoległość przetwarzania danych jest standardem dla profesjonalnych aplikacji komputerowych, gdyż są one w stanie wykorzystać potencjał, który zapewnia wielordzeniowość dzisiejszych komputerów

Propozycją wykorzystania równoległości jest architektura *„Multiple Instructions, Single Data”.*



Ilustracja 1 Schemat działania algorytmu.

Wybór tej metody uzasadniam poprzez charakter projektowanej aplikacji, która analizuje w danym czasie wyłącznie jeden plik, stąd też nie możemy zastosować architektury *„Single Instruction, Multiple Data”*, która jest częściej spotykanym podejściem.

Algorytmy analizy sygnału w czasie, jak i w dziedzinie częstotliwości dokonują operacji i przekształceń wszystkich próbek, stąd też w większości mają te same czasy potrzebne na ich wykonanie. Dzięki zrównolegleniu uzyskujemy zamiast $x[s] \cdot n$ czasu analizy, po prostu $x[s]$.

Przykład kodu zgodnego z powyższymi założeniami, realizującego równoległą parametryzację:

```
    for (i, [param, func]) in enumerate(function_dict.items()):  # execute each function from dict

        print(i,param, func)

        pool.apply_async(extract_attributes, args=(i, func, chunks, tot, param))

    pool.close()
    pool.join()
```

Dla słownika *function_dict*, wszystkie jego przedmioty (*ang. items*), czyli funkcje wykonywane są na sygnale, ale w osobnych wątkach wywoływanych za pomocą instrukcji *pool.apply_async()* powodujące asynchroniczne wykonanie funkcji parametryzującej.

```
function_dict = {
    "SR": spectral_rolloff,
    "SF": spectral_flatness,
    "SC": spectral_centroid,
    "MFCC": get_mfcc,
    "EIB": energy_in_bands,
    "Peaks": get_peak,
    "SNR": get_snr,
    "RMS": rootmean,
    "ZC": zero_cross

}
```

Powyżej zaprezentowano przykład słownika zawierającego spis funkcji, które należy wykonać na sygnale. Nazwa funkcji w języku Python jest wskaźnikiem na adres tej funkcji, co umożliwia wywoływanie w pętli różnej funkcji przy tej samej zmiennej.

```
def extract_attributes(pron_num, f, data, params, p_name): #, rdy):
    print('Created separated process for: ', p_name)
    start = time.time()
    temp = []
    for chunk in data:  # for each chunk of data
        temp.append(f(chunk))
    params[pron_num]=temp
    print('process ended for: %s in %0.2fs' % (p_name, time.time()-start))
```

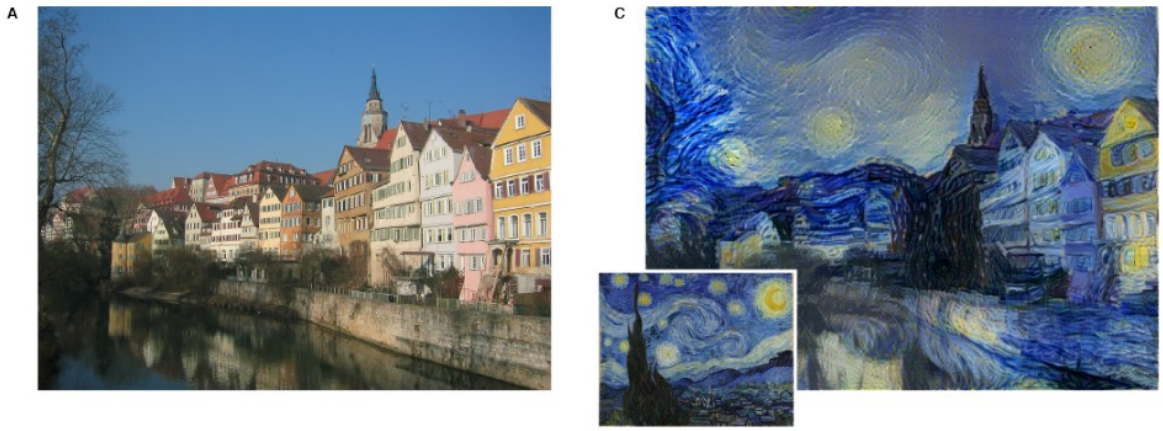
Ostatnim elementem jest sama funkcja bezpośrednio wywoływana w pętli i wykonująca daną instrukcję ze słownika funkcji parametryzującej. Pełni ona rolę „Wrappera” poprzez iteracyjne wykonywanie analizy i zapamiętywanie wyniku w tablicy i ostatecznie zapisywanie do pamięci współdzielonej reprezentowanej poprzez zmienną *params*.

4. Uczenie maszynowe – Neural Networks

[napisać jakiś wstęp teoretyczny o tym jak działa uczenie maszynowe]

W celach eksperymentalnych rozważono wykorzystanie algorytmów uczenia maszynowego do wcześniej omówionego programu. Uczenie maszynowe pozwala na optymalizację oraz znajdywanie rozwiązań dla złożonych problemów na podstawie danych wejściowych stanowiących zbiór danych treningowych algorytmu

Pomysł ten powstał dzięki lekturze (1) „A Neural Algorithm of Artistic Style", która opisuje użycie algorytmów Deep Neural Network do ekstrakcji cech obrazu składających się na jego styl artystyczny, jaki jest postrzegany przez człowieka. Jako dane wejściowe użyte zostają trzy obrazy. Jeden stanowi obiekt poddawany transformacji, z drugiego pobierane są dane o stylu obrazu, a trzeci jest zawartością, do której dostosowany zostanie pierwszy obraz. Poniżej przedstawiono przykładowe działanie programu, gdzie danymi wejściowymi są dwa obrazy, przy czym pierwszy obraz został użyty zarówno jako ten, który ma zostać poddany transformacji, jak i ten reprezentujący zawartość. Obraz, z którego pobrano informacje o stylu znajduje się w lewym dolnym rogu obrazu „C".



Ilustracja 2 Przykładowy wynik działania omawianego algorytmu w „A Neural Algorithm of Artistic Style" (1).

[Analogiczne działanie Reinforced neural network, gdzie dane wejściowe to czyste nagranie i styl=odpowiedź impulsowa pomieszczenia.]

[może opis jak działa powyższy przykład i jakby mógł zostać zmodyfikowany do analizy spektrogramów -prosto – schemat/ pomysł]

[Myślę o napisaniu trochę do a propos pracy, którą znalazłem dotyczącej sieci neuronowej i detekcji ech/ analizy spektrogramów]

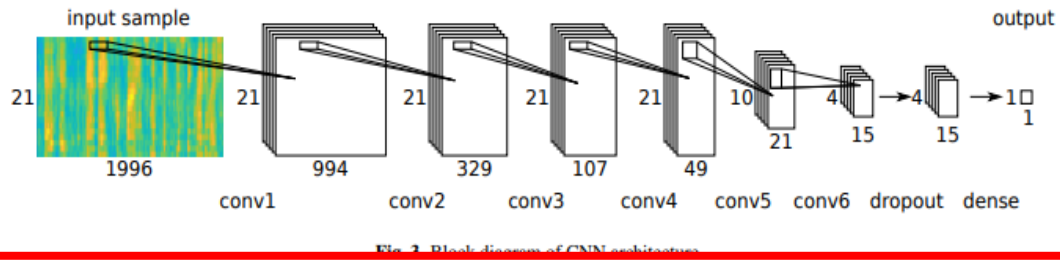


Fig. 3. Block diagram of CNN architecture

Działanie – funkcja optymalizująca -reward function /

on a white noise image to find another image that matches the feature responses of the original image. So let $\vec{p}$ and $\vec{x}$ be the original image and the image that is generated and P^l and F^l their respective feature representation in layer l. We then define the squared-error loss between the two feature representations

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^l - P_{ij}^l\right)^2 . \tag{1}$$

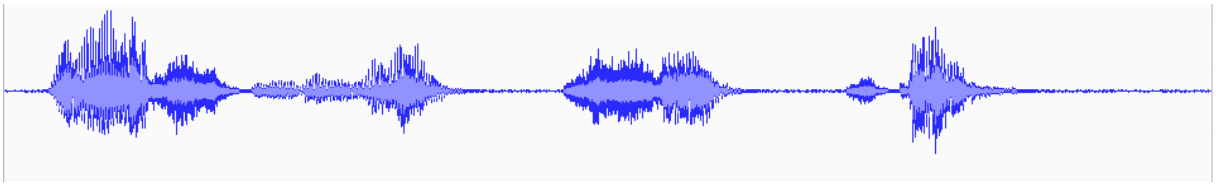
The derivative of this loss with respect to the activations in layer l equals

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^l} = \begin{cases} \left(F^l - P^l\right)_{ij} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 . \end{cases} \tag{2}$$

from which the gradient with respect to the image $\vec{x}$ can be computed using standard error back-propagation. Thus we can change the initially random image $\vec{x}$ until it generates the same response in a certain layer of the CNN as the original image $\vec{p}$. The five content reconstructions in Fig 1 are from layers 'conv1_1' (a), 'conv2_1' (b), 'conv3_1' (c), 'conv4_1' (d) and 'conv5_1' (e) of the original VGG-Network.

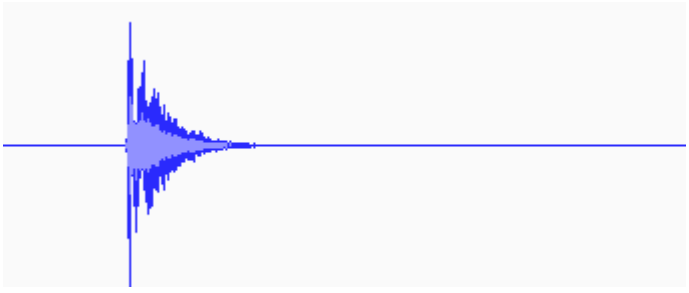
5. Podsumowanie

Rezultatem działania zaprojektowanej aplikacji są dane przetworzonych dwóch nagrań audio stanowiących dane wejściowe programu. Aplikacja dokonuje analizy nagrania odpowiedzi impulsowej danego pomieszczenia wyznaczając czas pogłosu oraz szacuje barwę nagrania rozumianą jako określenie środka ciężkości charakterystyki częstotliwościowej, wskazując w ten sposób sparametryzowany współczynnik wysokości dominujących częstotliwości w nagraniu, przez co możemy określić wpływ pomieszczenia na nagranie (przykładowo wokal) oraz względną zmianę barwy w odniesieniu do innej próbki. Następnie na podstawie próbki odpowiedzi impulsowej, możemy nałożyć jej charakterystykę, na czyste nagranie studyjne poprzez splot. Poniżej przedstawiono przykłady:

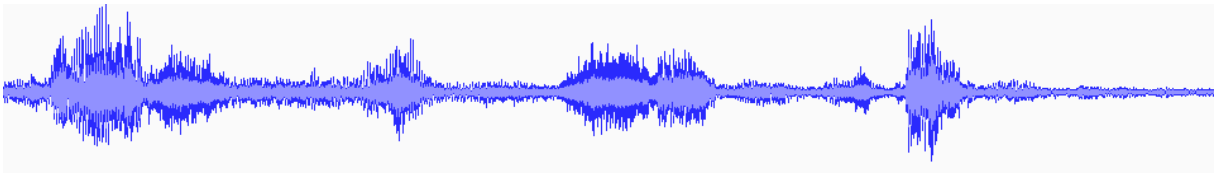


Ilustracja 3 Kształt fali przedstawiającej plik wejściowy, będący nagraniem mowy.

Na powyżej przedstawione nagranie nałożono odpowiedź impulsową pomieszczenia zaprezentowaną na ilustracji poniżej.

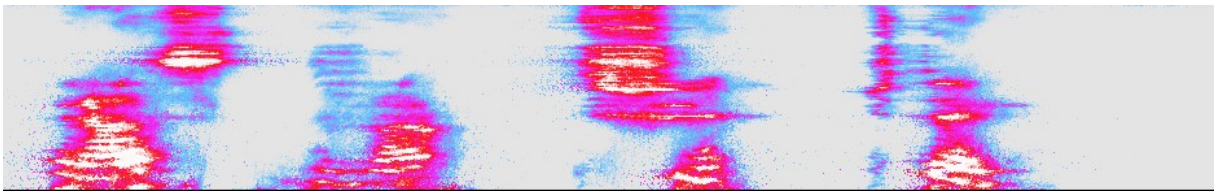


Ilustracja 4 Kształt fali - drugi plik wejściowy - odpowiedź impulsowa pomieszczenia.

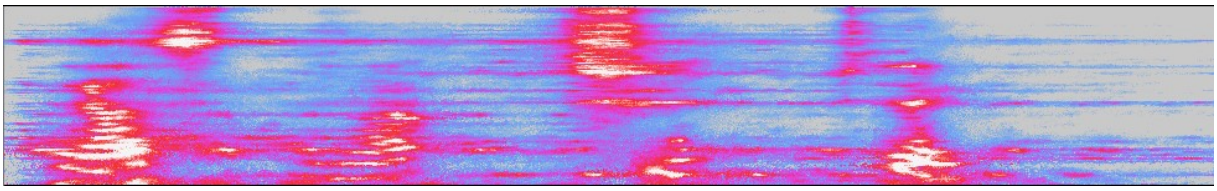


Ilustracja 5 Kształt fali - plik wyjściowy.

Na pliku stanowiącym rezultat działania programu zaprezentowany na ilustracji powyżej wyraźnie widać zwiększony czas pogłosu, co na spektrogramie widoczne będzie jako rozmycie w czasie.



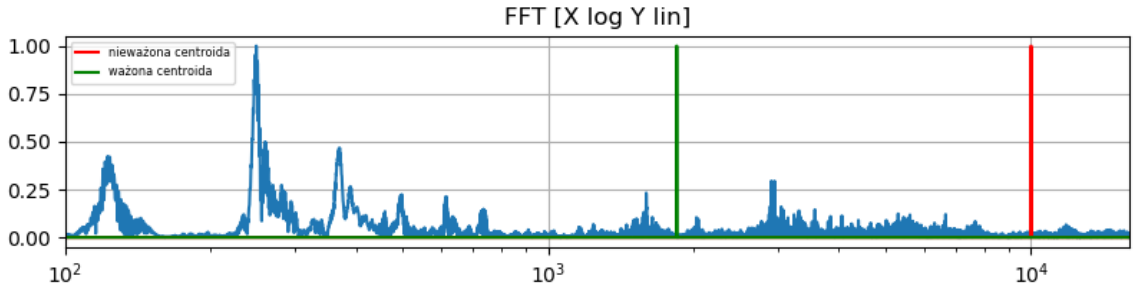
Ilustracja 6 Spektrogram - plik wejściowy - nagranie mowy.



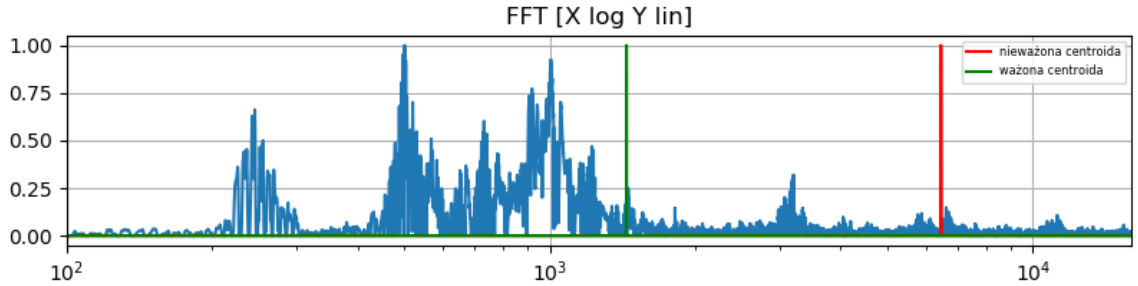
Ilustracja 7 Spektrogram - plik wyjściowy.

Korekta nagrania do wzorca za pomocą splotu odpowiedzi impulsowej jest znacznie łatwiejszym zagadnieniem niż operacja do tego odwrotna, która nie sprawdza się w teorii dając jedynie efekt dzwonienia będącego wynikiem wzmocnienia sygnału zawierającego szumy. Tej części pracy nie udało się zrealizować ze względu na nieakceptowalne wyniki działania zaimplementowanego algorytmu rozplotu (pod względem efektu działania w kontekście edycji nagrania dźwiękowego).

Wydłużenie czasu pogłosu, a także nadanie nagraniu charakterystyki wybrzmienia pokoju powinno spowodować zmianę barwy na cieplejszą, co możemy przeanalizować dzięki implementacji algorytmu spektralnej centroidy. Na wykresach przedstawiono wyniki działania oryginalnej [na czerwono] oraz zmodyfikowanej [na zielono] wersji tego algorytmu.



Wykres 9 Średnie FFT (całego) oryginalnego nagrania przed poddaniem obróbce przez algorytm.



Wykres 10 Średnie FFT (całego nagrania) po obróbce przez algorytm.

Jak widać, zgodnie z oczekiwaniami wzrósł udział niskich częstotliwości (poniżej 1kHz) poprzez wprowadzenie wpływu akustyki pomieszczenia, co więcej częstotliwości utrzymywały się przez dłuższy okres czasu przez wzrost czasu pogłosu.

Badając charakter nagrania audio w dziedzinie czasu najważniejszymi elementami mającymi wpływ na jego brzmienie są przede wszystkim odbicia powstające w pomieszczeniu, w którym wykonano nagranie, pierwsze odbicia oraz stosunek energii pierwszych 50ms do reszty ogona pogłosowego nagrania impulsu w pomieszczeniu. W dziedzinie częstotliwości natomiast zaobserwować możemy zmianę barwy dźwięku przez akustykę pomieszczenia, czy też rezonanse powstające przez geometrię pomieszczenia.

Parametryzacja wyżej omówionych elementów nagrania pozwala na implementację algorytmów uczenia maszynowego, które pozwolą na dokonanie eksperymentalnych korekt nagrania celem zmiany brzmienia nagrania czy usuwania wpływu pomieszczenia na nagranie, gdyż zastosowanie metody rozplotu odpowiedzi impulsowej z nagrania wykonanego w pomieszczeniu okazała się wysoce niesatysfakcjonującą.

6. Bibliografia

1. **Leon A. Gatys Alexander S. Ecker, Matthias Bethge.** *A Neural Algorithm of Artistic Style.* 2015.

2. **Das Vinu, Ariwa, Ezendu, Rahayu, Syarifah Bahiyah.** *Signal Processing and Information Technology.*

3. **PN-EN ISO 3382.**

4. Everest F. Alton. ***Podręcznik Akustyki.*** **2013.**

5. acustica.ing.unibo.it. [Online] http://acustica.ing.unibo.it/Researches/room/convolution.html.

6. sciencedirect. [Online] https://www.sciencedirect.com/topics/neuroscience/deconvolution.

7. Hannes, Gamper and J., Tashev Ivan. ***BLIND REVERBERATION TIME ESTIMATION USING A CONVOLUTIONAL NEURAL.*** **Redmond, WA, USA : s.n.**

8. matplotlib. [Online] https://matplotlib.org.

9. scipy. [Online] https://www.scipy.org.

7. Wykresy/Tabele/Rysunki