Implementacja i omówienie algorytmów analizujących dźwięki z perspektywy porównywania utworów

(Implementation and analysis of audio processing algorithms in recording comparison approach)

Alicja Danilczuk, Klaudia Osowska

Praca inżynierska

Promotor: doktor Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

4 lutego 2021

Streszczenie

Poniższa praca „Implementacja i omówienie algorytmów analizujących dźwięki z perspektywy porównywania utworów" zajmuje się zagadnieniem znajdowania podobieństw między nagraniami. Celem pracy była analiza algorytmów wykorzystywanych w procesie oceny podobieństwa nagrań. Struktura pracy opiera się na czterech głównych częściach.

Pierwsze trzy oscylują wokół analizy i implementacji użytecznych algorytmów, służących do kolejno: znajdowania początków dźwięków w nagraniu, wyliczania wysokości odnalezionych dźwięków i dopasowywania melodii nagrania do pozostałych posiadanych danych. Do rozwiązania problemu znajdowania początków dźwięków zaproponowano Magnitude method, Short-term energy method, Surf method i Envelope match filter. Do wyliczania wysokości dźwięków wybrano Funkcję autokorelacji, Funkcję różnicy średnich wartości, Harmonic product spectrum, Trzecią największa wartość FFT oraz Metodę cepstralną. Programowanie dynamiczne i skalowanie liniowe zostało natomiast użyte do dopasowywania melodii. Każdy podrozdział opisujący algorytm posiada opis, schemat działania, uwagi odnośnie implementacji oraz rozpatrzenie działania na przykładowym nagraniu.

Czwarta część jest szerszym przeglądem testów wykonanych dla omawianych algorytmów. Przedstawiono porównanie ich wyników na każdym z trzech etapów analizy i omówiono otrzymane i oczekiwane wyniki.

The subject of the „Implementation and analysis of audio processing algorithms in recording comparison approach" thesis is an analysis of algorithms useful for comparing audio records in order to calculate their similarity. This thesis is divided into four parts.

The first three chapters revolve around the analysis and implementation of algorithms useful for onsets detection in the recording, calculating the pitch of the sounds, and matching the melody of the recording with the data from a data set. For onset detection discussed algorithms are Magnitude method, Short-term energy method, Surf method, and Envelope match filter. Algorithms presented for pitch extraction are: Autocorrelation function, Average magnitude difference function, Harmonic product spectrum, Third largest value of FFT, and Cepstral method. For melody matching there are two analyzed algorithms: Dynamic programming and Linear scaling. Each section consists the algorithm's description, implementation notes, and results for some example audio record.

The fourth section is an overview of the tests performed for the algorithms. A comparison of the results at each of the three stages of the analysis is presented and the obtained and expected results are briefly discussed.

Spis treści

Rozdział 1.

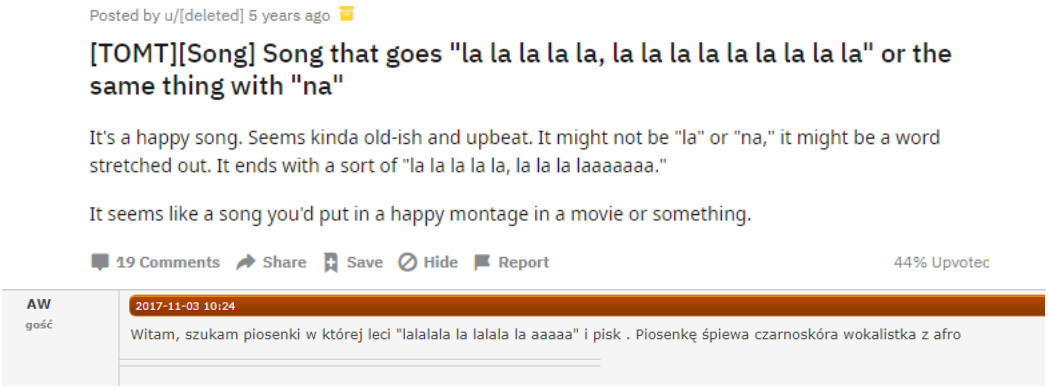
Wprowadzenie

Większość z nas choć raz spotkała się z sytuacją, w której chcieliśmy odszukać kiedyś usłyszaną piosenkę. Zadanie jest banalnie proste, gdy pamiętamy fragment tekstu – wpisanie go w przeglądarkę internetową zajmuje chwilę i daje bardzo dobre efekty nawet przy nie do końca dobrze zapamiętanych słowach. Algorytmy wyszukiwarek spełniają tu swoją rolę. Jeżeli szukana piosenka jest akurat odtwarzana, przyłożenie telefonu do źródła dźwięku wraz z uruchomioną aplikacją Shazaam[1] zajmuje tylko kilka sekund, a w rezultacie otrzymujemy nazwę, wykonawcę i okładkę albumu powiązane z piosenką. Nawet obrazki potrafimy znajdować na podstawie podobieństw – wyszukiwanie obrazem w Google[2] z roku na rok działa coraz lepiej. A co w przypadku, gdy znamy tylko melodię? Gdy obok siebie mamy osobę, która może kojarzyć ten utwór, pierwszym odruchem jest zanucenie fragmentu – początku, refrenu, najbardziej popularnej części. Jeżeli osoba nie zna tytułu nuconej piosenki lub nie mamy komu zanucić, to trzeba szukać innych rozwiązań. W internecie można znaleźć przykłady poszukiwań prowadzonych przez ludzi, którzy znaleźli się w takiej sytuacji (Rysunek 1.1). Jak łatwo się domyślić, odpowiadający mieli problem z udzieleniem pomocy. Po wpisaniu w wyszukiwarkę pytania „Jaka to piosenka" dostajemy informację o udostępnionej w 2020 roku przez Google[3] funkcji zanucenia lub zaśpiewania do telefonu by uzyskać odpowiedź. I funkcja nawet nieźle działa... dla angielskojęzycznych utworów oraz tych posiadających unikalne, łatwo rozpoznawalne fragmenty. Piosenka Krzysztofa Krawczyka „Chciałem być" okazała się dla algorytmu nie do przeskoczenia. Jednakże, skoro dla pewnej grupy utworów funkcjonalność działa, to można się spodziewać, że powstało wiele algorytmów, które pomagają rozwiązać ten problem.

[1] `https://www.shazam.com/pl`

[2] `https://www.google.pl/imghp?hl=pl`

[3] `https://blog.google/products/search/hum-to-search/`



Rysunek 1.1: Poszukiwania tytułu piosenki w internecie. [11] [12]

W kolejnych rozdziałach tej pracy przedstawimy kilka z takich algorytmów, użytecznych w kolejnych krokach analizy sygnału dźwiękowego. Zaprezentujemy opis ich działania, a następnie pokażemy jak działają w praktyce opierając się na naszych implementacjach. Będziemy patrzeć na ścieżki dźwiękowe i przeszukiwać bazę w celu znalezienia utworu, który mógł być pierwowzorem zaśpiewanej lub zanuconej ścieżki. Chcemy znaleźć odpowiedź na pytanie „Jaką piosenkę zanucono/zaśpiewano?", czyli które nagranie z bazy danych jest najlepszym dopasowaniem do naszego wykonania. W tym celu najpierw przyjrzymy się czterem algorytmom znajdującym w nagraniu początki dźwięków. Mając określone owe początki, przejdziemy do kolejnych pięciu algorytmów znajdujących wysokości występujących dźwięków, dzięki czemu będziemy mogli stwierdzić jaka nuta odpowiada konkretnemu dźwiękowi. Ostatnim krokiem będzie przegląd algorytmów dopasowujących melodię naszego nagrania do ustalonej piosenki z bazy i stwierdzających na ile te dwa nagrania są do siebie podobne. Aby znaleźć najlepszą drogę do dopasowania utworów na każdym etapie analizy dźwięków wprowadzimy i przetestujemy kilka rozwiązań, z których wybierzemy subiektywnie najlepsze do wykorzystania jako baza kolejnego kroku.

Na każdym etapie rozważymy, który algorytm wydaje się podawać najdokładniejsze wyniki i jest najbardziej wiarygodny, oraz zastanowimy się czy po połączeniu kilku algorytmów lub lekkiej modyfikacji uzyskamy lepsze wyniki.

Wszystkie algorytmy zostały zaimplementowane w języku Python. Analizowane nagranie musi znajdować się w pliku *.wav*, wczytywane jest przy pomocy biblioteki *Librosa*.

Algorytmy przedstawione w tej pracy są w dużej mierze oparte na publikacji „Query by Singing and Humming" napisanym przez Chiao-Wei Lin. [13].

Rozdział 2.

Znajdowanie położenia dźwięków w czasie

Pierwszym krokiem analizy sygnału dźwiękowego będzie znalezienie kolejnych występujących w nim pojedynczych dźwięków. Początki szukanych dźwięków, dla zgrabności opisów, czasami będziemy nazywać słowem *onset* (z angielskiego: nadejście, początek). Każdemu dźwiękowi odpowiada jedna nuta. Dobry algorytm znajdujący położenie dźwięków w czasie powinien spełniać dwie własności:

1. Trafność, dokładność – jeśli w okolicach miejsca n_1 w nagraniu znajduje się początek dźwięku, to algorytm powinien je podać. Jeśli algorytm poda początek dźwięku w okolicach miejsca n_2, to faktycznie w nagraniu początek powinien tam występować.
2. Wydajność - czas obliczania położenia dźwięków powinien być nieduży.

Przeanalizujemy cztery algorytmy opisane w pracy Query by singing and humming [13]. Każdy z nich przetestujemy na kilku nagraniach, a na koniec porównamy wyniki. Podamy również metodę, która może zostać użyta, jeżeli żaden z algorytmów nie zwraca satysfakcjonujących nas wyników.

2.1. Magnitude method

2.1.1. Przedstawienie algorytmu

W wolnym tłumaczeniu Metoda wielkościowa. Jest bardzo intuicyjna, polega na znalezieniu początków dzwięków kierując się różnicami głośności. W poniższym algorytmie będziemy analizować sygnał przepuszczony przez LPF, czyli przez filtr dolnoprzepustowy (ang. *Low-pass filter*). W każdym kolejnym oknie znajdujemy maksymalną wartość przefiltrowanego sygnału i zapisujemy w tablicy A. Następnie

sprawdzamy czy różnica wartości między elementem a jego poprzednikiem w A jest większa od jakiegoś ustalonego progu (*threshold*). Jeżeli jest większa to początek tego okna uznajemy za początek dźwięku.

Filtr dolnoprzepustowy to element przetwarzający sygnał, w naszym przypadku algorytm, który przepuszcza częstotliwości sygnału poniżej ustalonej częstotliwości granicznej, a tłumi składowe widma leżące w górnej jego części. Użyjemy filtru Chebyszewa typu 2 wbudowanego w bibliotekę scipy.signal.

Przebieg algorytmu:

1. $A_k = \max(LPF\{x[n]\}|kn_0 \leq n \leq (k+1)n_0)$
   gdzie x jest sygnałem wejściowym, n_0 wielkością okna a LPF to filtr dolnoprzepustowy
2. $D_k = A_k - A_{k-1}$
3. Jeżeli $D_k > threshold$, to kn_0 jest wyliczoną pozycją początkową dźwięku

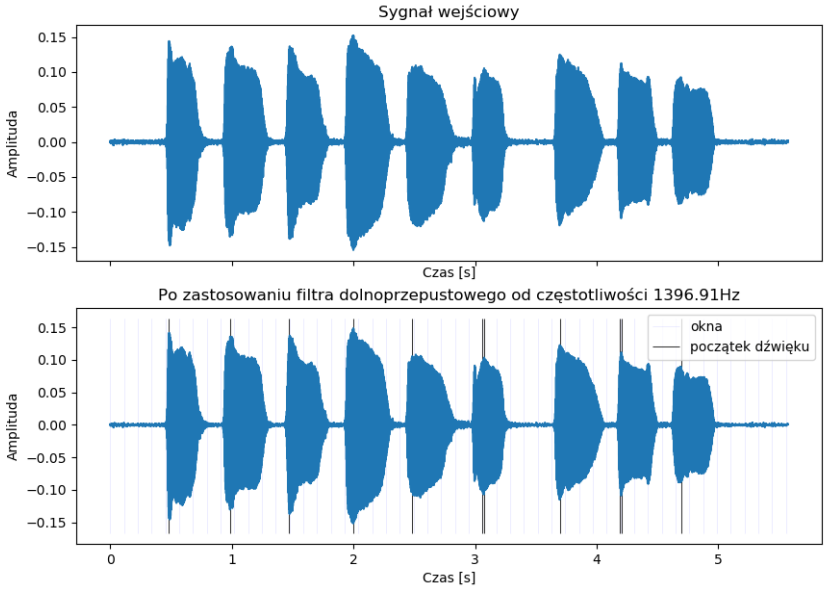
Aby uzyskać poprawne wyniki należy wybrać odpowiednie stałe: wielkość okna n_0 i próg *threshold*. Są dwa sposoby znajdowania ich. Pierwszym jest dobranie wielkości okna w taki sposób, by w większości obejmowało jeden dźwięk. Wtedy maksimum okien pobocznych będzie wystarczająco małe by różnica była większa od progu. Drugim może być wybranie na tyle małego okna, aby to głównie od progu zależały znalezione dźwięki. Musi on być jednak na tyle mały, by drobne wahania głośności nie dodawały kolejnych onsetów.

Wybór wartości n_0 i *threshold* przy których algorytm będzie zwracał prawidłowe wyniki dla dowolnego nagrania jest trudnym zadaniem. Z tego powodu dla każdego przeprowadzonego testu, wartości tych parametrów zostały dobrane ręcznie. Chcieliśmy aby wynik dla konkretnej badanej piosenki wynik był jak najlepszy (jak najwięcej prawidłowo podanych onsetów oraz jak najmniej niepodanych lub podanych w miejscach, gdzie nie powinny zostać zwrócone).

2.1.2. Analiza wyników

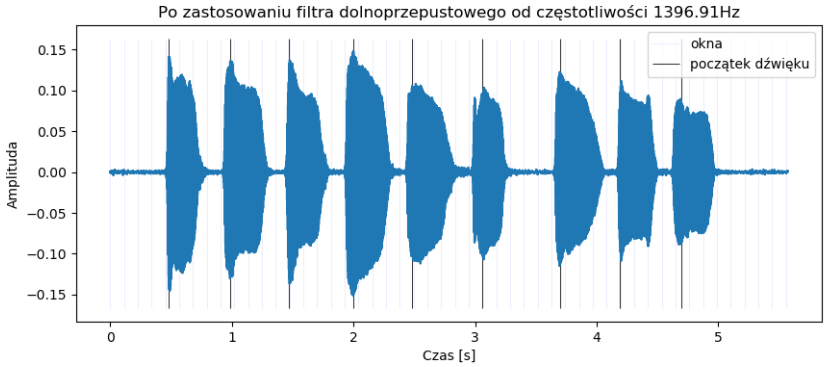
Do przetestowania działania opisywanego algorytmu użyjemy dwóch znacząco różniących się nagrań. Pierwszego, w którym kolejne dźwięki rozpoczynają się najgłośniej i drugiego, w którym nagrano nucenie w bardziej spokojny sposób.

Pierwszy sposób nucenia „Wlazł kotek na płotek" pokazany został na Rysunku 2.1. Pionowe czarne linie oznaczają znalezione przez program początki dźwięków. W tym nagraniu, przy wyborze stosunkowo niedużych okien już na początku dostajemy dosyć poprawne wyniki. Widać, że przy odpowiednio stromym początku dźwięku wielkość progu nie ma dużego wpływu na rezultaty – amplituda głośności jest duża więc o ile odpowiednio znaczna część początku zmieści się w oknie to onsety



Rysunek 2.1: „Wlazł kotek na płotek v.1", Magnitude method $n_0 = 2500, threshold = 0.006$

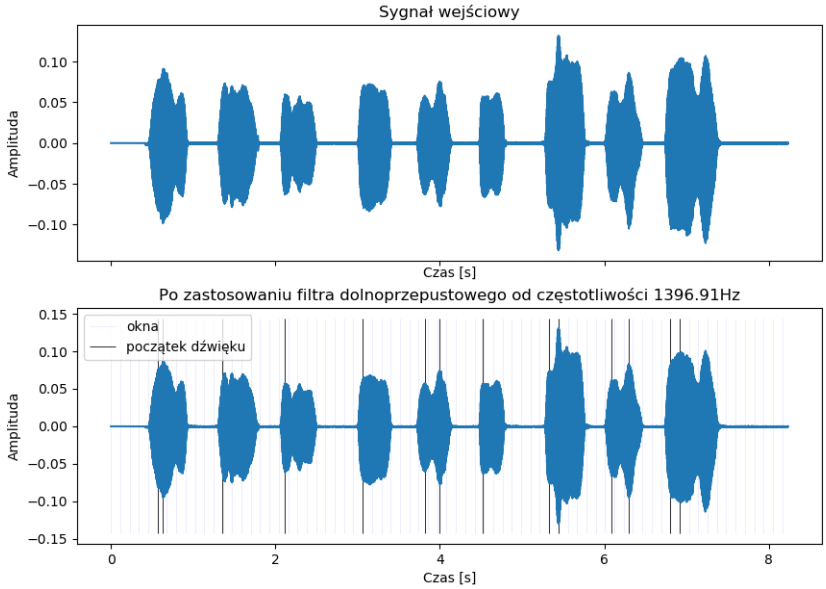
zostaną zaznaczone poprawnie. Oczywiście wciąż pojawiają się nieścisłości w ilości znalezionych onsetów. Poprawimy je podwyższając próg do 0.015 (Rysunek 2.2).



Rysunek 2.2: „Wlazł kotek na płotek v.1", Magnitude method $n_0 = 2500, threshold = 0.015$

Po podwyższeniu progu kolejne dźwięki zostają poprawnie zaznaczone. Przyjrzyjmy się teraz jak algorytm zachowuje się przy nagraniu z zaokrąglonymi początkami dźwięków. Ten drugi sposób nucenia pokazano na Rysunku 2.3.

Widzimy, że tutaj znajdowane są wszystkie początki, niestety zaznaczeń jest więcej niż oczekiwano. Sugeruje to zwiększenie progu, jednak po tym zabiegu możemy stracić znalezione już, rzeczywiste początki, jeżeli nachylenie wykresu nie było zbyt duże. Szybko widać, że metoda ta jest bardzo podatna na zmiany wartości

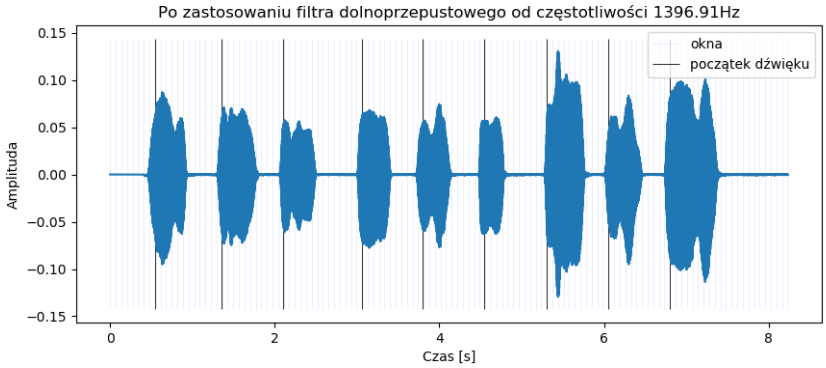


Rysunek 2.3: „Wlazł kotek na płotek v.2", Magnitude method $n_0 = 2500, threshold = 0.015$

stałych i tylko dokładne dopasowanie ich da doskonałe wyniki w konkretnych przypadkach. Niestety, jest też w dużym stopniu podatna na błędy obliczeń w sytuacjach, gdy wartości głośności w kolejnych oknach są bliskie sobie oraz gdy okna ułożą się w niefortunnych miejscach i w niepożądany sposób przetną początek dźwięku.

Po wykonaniu paru testów powyższej wersji algorytmu, do programu zostało dodane usprawnienie, które zwiększa dokładność wyników. Zmniejszamy okno pozwalając na znajdowanie większej liczby początków niż powinno zostać zwrócone. Następnie po obliczeniu wszystkich początków przeglądamy je od najwcześniejszego i dla każdego początku p usuwamy wszystkie początki k, które są na prawo od p oraz w odległości od niego nie większej niż pewien z góry ustalony *epsilon*.

Po zastosowaniu tego usprawnienia z *epsilon* równym 6000 dostajemy wynik przedstawiony na rysunku 2.4.



Rysunek 2.4: „Wlazł kotek na płotek v.2", Magnitude method $n_0 = 1500, threshold = 0.015$, $\epsilon = 6000$

2.2. Short-term energy method

2.2.1. Przedstawienie algorytmu

Short-term energy method zakłada, że pomiędzy dwoma sąsiednimi nutami zawsze występuje fragment ciszy. Jest to podobna metoda do Magnitude method, jednak w jej przypadku zamiast stosować dla każdego okna filtr dolnoprzepustowy i wybierać maksymalny element w oknie, obliczamy dla niego poniżej zdefiniowaną energię. Wzór na energię, który stosujemy wygląda następująco:

1. $E_k = \sum_{n=kn_0}^{(k+1)n_0-1} x^2[n]$

   Gdzie n_0 oznacza wielkość okna

2. $D_k = E_k - E_{k-1}$

3. Jeżeli $D_k > threshold$, to na pozycji kn_0 występuje początek dźwięku.

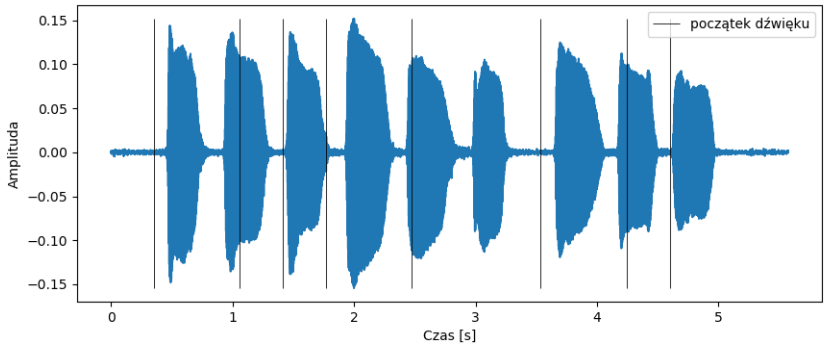
Najpierw dla każdego okna obliczamy energię, następnie już tak jak w Magnitude method obliczamy dla wszystkich sąsiednich okien różnicę policzonych dla nich wartości. Jeśli różnica ta jest większa niż pewien wcześniej ustalony próg($threshold$), to stwierdzamy, że znaleźliśmy położenie początku dźwięku.

2.2.2. Analiza wyników

Problemem znowu, jak w poprzedniej metodzie, jest dobranie odpowiedniego rozmiaru okna oraz progu. Jeśli próg będzie zbyt niski, to program będzie znajdował miejsca, które nie są początkiem nowej nuty. Metoda ta jest również wrażliwa na szumy. Poniżej przedstawimy kilka testów.

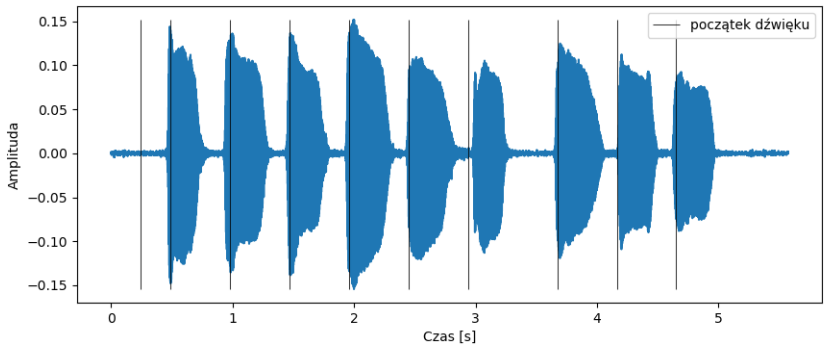
Pierwszym testem było sprawdzenie wyników dla około pięciosekundowego nagrania piosenki „Wlazł kotek na płotek” wykorzystywanego już przy analizie poprzedniego algorytmu. W tle nagrania nie występują żadne dodatkowe dźwięki ani szumy.

Dla progu równego 0.2 i rozmiaru okna 7800 program implementujący ten algorytm daje wynik pokazany na Rysunku 2.5.



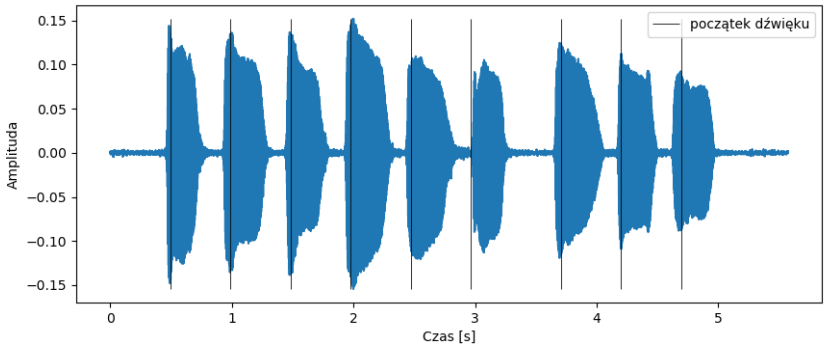
Rysunek 2.5: „Wlazł kotek na płotek v.1”, Short-term energy method $n_0 = 7800, threshold = 0.2$

Jak widać jedne z nich są bliżej miejsc w których powinny wystąpić, a jedne są dalej. Jednak w pobliżu niektórych początków dźwięków program nie podał żadnej odpowiedzi, na przykład w przypadku dźwięku występującego w okolicach trzeciej sekundy. Po zmniejszeniu okna do 5400 i zwiększeniu progu do 2 otrzymujemy rezultat z Rysunku 2.6. Wynik jest o wiele lepszy, jednak pierwszy początek jest podany nadmiarowo.



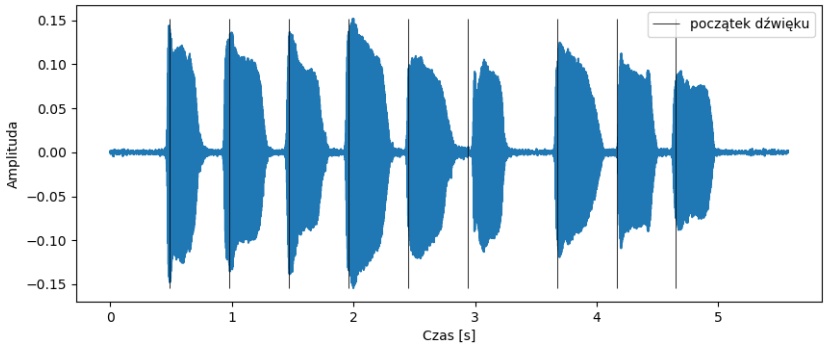
Rysunek 2.6: „Wlazł kotek na płotek v.1”, Short-term energy method $n_0 = 5400, threshold = 2$

Okazuje się, że dla okna 5450 oraz progu 5 program dla tego nagrania zwraca bardzo dobry wynik (Rysunek 2.7).



Rysunek 2.7: „Wlazł kotek na płotek v.1", Short-term energy method $n_0 = 5450, threshold = 5$

Później program został uruchomiony po dodaniu do niego usprawnienia opisanego w Magnitude method, czyli na sam koniec działania usuwania początków, które mają na lewo od siebie inny początek w odległości co najwyżej *epsilon*. Dla progu równego 5, rozmiaru okna 5400 oraz epsilon 8000 program implementujący ten algorytm zwraca zaznaczenia zaprezentowane na Rysunku 2.8.



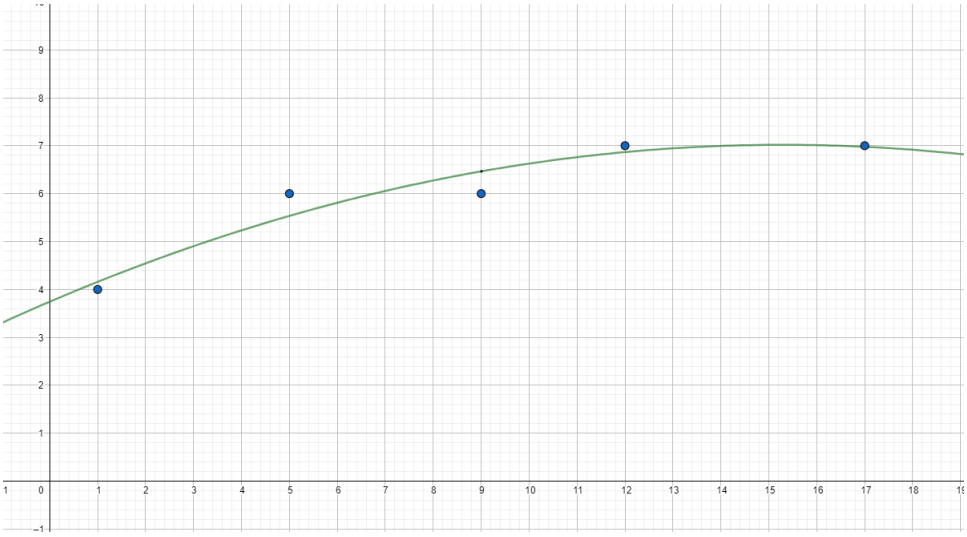
Rysunek 2.8: „Wlazł kotek na płotek v.1", Short-term energy method $n_0 = 5400, threshold = 5, \epsilon = 8000$

Optymalny próg jest znacznie większy niż w poprzednio opisanej metodzie Magnitude method. Najprawdopodobniej wynika to z tego, że zamiast brać bezpośrednie wartości sygnału wejściowego bierzemy pod uwagę ich kwadraty.

2.3. Surf method

2.3.1. Przedstawienie algorytmu

Metoda surf polega na obliczaniu nachylenia przez dopasowanie funkcją wielomianu drugiego rzędu. Chcemy wyliczyć jaka w przybliżeniu byłaby wartość w punkcie, gdyby wziąć pięć kolejnych wartości i zaśpiewać je idealnie.



Rysunek 2.9: Surf method – Styczna

Przebieg algorytmu:

1. $A_k = \max(x[n] | kn_0 \leq n \leq (k+1)n_0)$

   gdzie x jest sygnałem wejściowym a n_0 wielkością okna

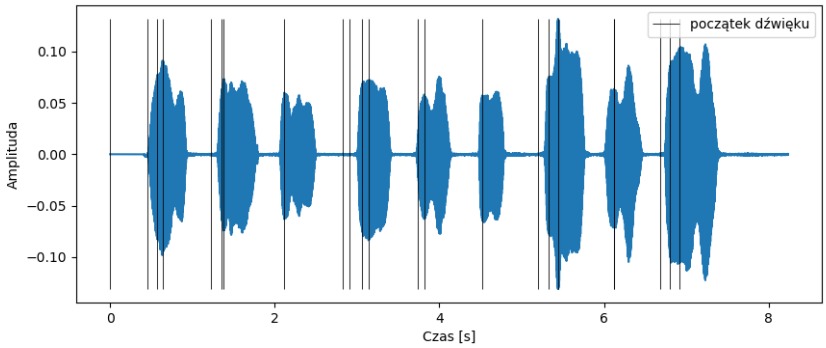
2. Przybliżamy A_m dla pięciu sąsiadujących elementów $m = (k-2 \sim k+2)$ przez funkcję wielomianu drugiego rzędu $p[m] = a_k + b_k(m-k) + c_k(m-k)^2$. Współczynnik b_k to nachylenie w środkowym punkcie, gdzie $(m = 0)$:

$$b_k = \sum_{\tau=-2}^{2} A_{k+\tau}\tau / \sum_{\tau=-2}^{2} \tau^2$$

3. Jeżeli $b_k > threshold$, to kn_0 jest wyliczoną pozycją początkową dźwięku

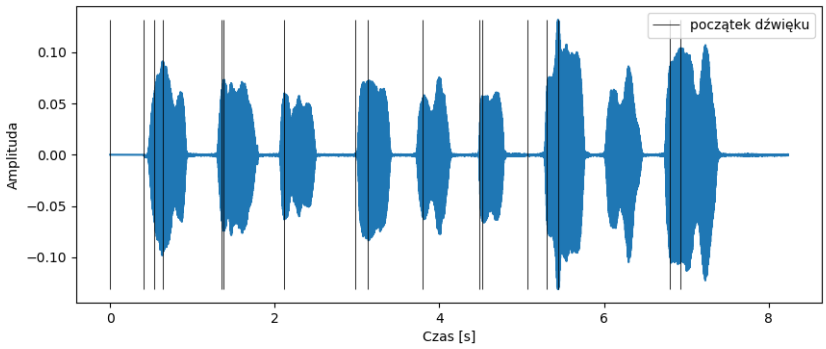
2.3.2. Analiza wyników

Przyjrzyjmy się teraz efektom działania algorytmu dla analizowanego już wcześniej nagrania. Ponieważ w obliczeniach bierzemy pod uwagę 5 kolejnych wartości, można wywnioskować, że najlepiej działać będzie okno stosunkowo małe.



Rysunek 2.10: „Wlazł kotek na płotek v.2", Surf method $n_0 = 2500, threshold = 0.01$

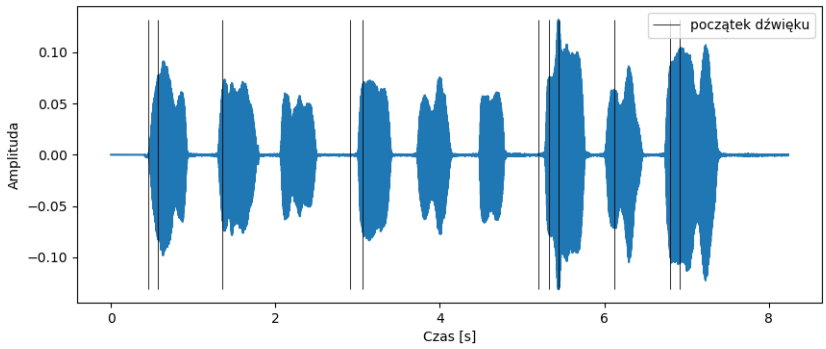
Rysunek 2.10 pokazuje, że powyższa metoda znajduje wszystkie początki dźwięków, niestety zaznacza też dodatkowe. Wynika to z faktu, że uwzględniając w obliczeniu 5 kolejnych wartości jest duża szansa, że sąsiadujące ze sobą elementy będą miały stosunkowo podobne wartości. Można w tej sytuacji zwiększyć rozmiar okna (Rysunek 2.12).



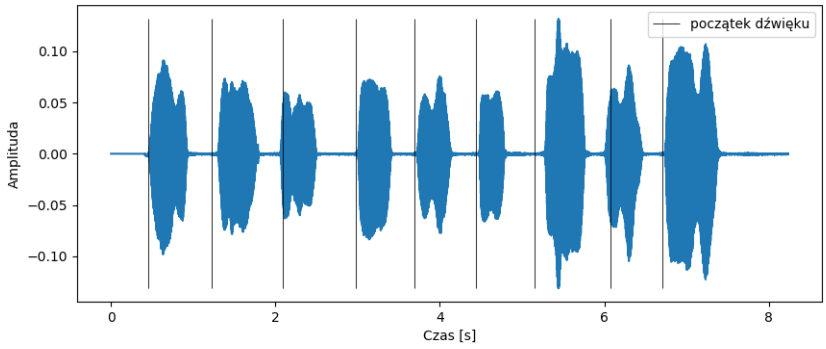
Rysunek 2.11: „Wlazł kotek na płotek v.2", Surf method $n_0 = 3000, threshold = 0.01$

Widać, że liczba nadprogramowych onsetów zmniejszyła się jednak straciliśmy też zaznaczenie jednego z dźwięków, czyli zmniejszenie okna nie pomaga w tym przypadku. Tak samo nie pomaga zwiększenie progu (Rysunek 2.12) – utraciliśmy zaznaczenie kilku początków jednocześnie niektóre wciąż zostały zaznaczone wielokrotnie. Pokazuje to jak bardzo powyższa metoda zależna jest od wejściowego sygnału.

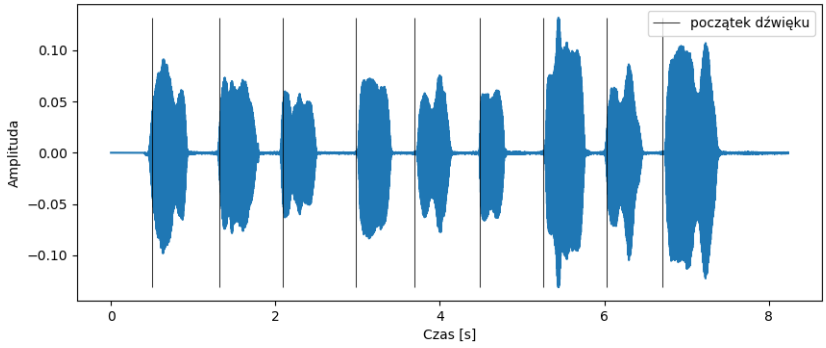
Po uruchomieniu powyższego algorytmu ze stosowanym już wcześniej usprawnieniem z ϵ wyniki są zdecydowanie lepsze, co widać na Rysunku 2.13. Aby precyzyjniej zaznaczyć początki dźwięków zmniejszymy okno jeszcze bardziej (Rysunek 2.14).



Rysunek 2.12: „Wlazł kotek na płotek v.2”, Surf method $n_0 = 2500, threshold = 0.02$



Rysunek 2.13: „Wlazł kotek na płotek v.2”, Surf method $n_0 = 2000, threshold = 0.015, \epsilon = 4000$



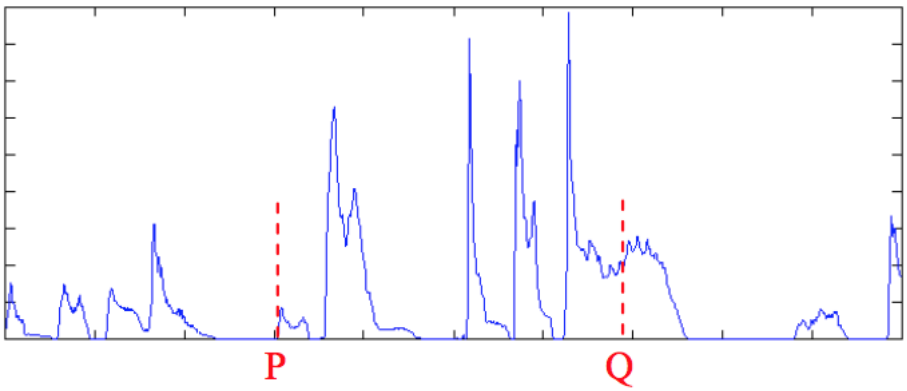
Rysunek 2.14: „Wlazł kotek na płotek v.2” – Surf method $n_0 = 1000, threshold = 0.015, \epsilon = 4000$

2.4. Envelope match filter

2.4.1. Przedstawienie algorytmu

Jak zostało wspomniane na początku tego rozdziału, dobry algorytm do znajdowania położenia początków dźwięków w czasie powinien cechować się wysoką dokładnością. Zatem nie może być on wrażliwy na szumy w tle (przykładowo na dźwięk wiertarki lub deszczu, które nie są częścią nucenia). Metoda powinna być również wydajna. W Envelope match filter zamiast obliczać dla każdego elementu w oknie wartość funkcji (jak filtr dolnoprzepustowy dla Magnitude method), znajdujemy jedynie największą wartość w danym oknie i to ją wykorzystujemy do dalszych obliczeń. Jednak chcąc poradzić sobie z błędnymi wynikami spowodowanymi szumem w tle chcielibyśmy skorzystać z pewnej funkcji filtrującej. Zamiast korzystać z filtra dolnoprzepustowego, w tej metodzie dla każdej maksymalnej wartości w oknie skorzystamy z filtra dopasowującego, który zostanie opisany niżej. Jest on w stanie dobrze zredukować wpływ szumu z tła na wyniki.

Zauważmy, że obliczanie różnic energii policzonych bezpośrednio na naszym nagraniu może powodować problemy. Rozważmy nagranie przedstawione na rysunku 2.15.



Rysunek 2.15: W miejscu P powinien zostać wykryty początek, ale w miejscu Q już nie [3].

Podążając za analizą przeprowadzoną w „Improved onset detection algorithm based on fractional power Envelope match filter" [3] możemy zauważyć, że w miejscu P oczywiście mamy początek dźwięku. W miejscu Q pojawiają się wahania wysokości dźwięku, ale nie zaczyna się w tym miejscu nowa nuta. Jednak różnica wysokości dla okna zaczynającego się w Q i okna tuż przed nim (oznaczmy je przez $Q-1$) jest nieco większa niż dla okna zaczynającego się w P i $P-1$ mimo, że miejsce Q nie powinno zostać podane jako początek dźwięku. Jest tak dlatego, że dźwięk w okolicach P jest cichy. Załóżmy, że $A_p = 0.1$, $A_{p-1} = 0$, $A_q = 0.52$, $A_{q-1} = 0.4$, gdzie A_k to wartość, którą zdefiniowaliśmy wcześniej w algorytmie Magnitude method, dodatkowo przez q oznaczymy okno w którym znajduje się element Q, podobnie dla

pozostałych. Teraz rozważając różnice sąsiednich okien, w miejscu Q mamy różnicę 0.12, a w miejscu P 0.1, więc przy pewnym progu, który wykrywa początki cichych dźwięków (nasze P), mogą zostać też podawane nieprawidłowe miejsca takie jak Q. Aby poradzić sobie z takimi sytuacjami, dla każdej amplitudy w naszym nagraniu policzymy jej wartość do pewnej potęgi λ takiej, że $0 < \lambda < 1$. Jeśli $\lambda = 0.7$, to po podniesieniu każdej z wartości A_p, A_{p-1}, A_q, A_{q-1} do potęgi λ, otrzymamy $A_p^\lambda - A_{p-1}^\lambda = 0.1^{0.7} - 0^{0.7} \approx 0.1995 > A_q^\lambda - A_{q-1}^\lambda = 0.52^{0.7} - 0.4^{0.7} \approx 0.1062$. Różnica tych dwóch wartości jest teraz większa.

Do dalszej analizy potrzebujemy definicji konwolucji (inaczej splotu). Operacja splotu funkcji z wartościami dyskretnymi jest następująca [7]:

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

Zauważmy, że w Magnitude method aby obliczyć różnice wartości otrzymanych dla kolejnych okien możemy równie dobrze policzyć konwolucję wartości A_k z filtrem [1, -1]. Przykładowo, jeśli tablica otrzymanych wartości A_k byłaby postaci [3, 6, 2, 7], to licząc jej konwolucję z filtrem [1,-1] otrzymamy [3, 6∗1+3∗(−1), 2∗1+6∗(−1), $7 * 1 + 2 * (-1)$]. Widać, że są to różnice kolejnych wartości.

W naszej metodzie zamiast obliczać konwolucję z filtrem [1,-1], użyjemy filtra dopasowującego. Idea jest następująca: chcielibyśmy mieć filtr, który jest w stanie jak najlepiej odfiltrować początki ludzkiego głosu od możliwych szumów w tle. Aby to zrobić potrzebowalibyśmy „szablonu", który idealnie opisuje szukany sygnał (okolice początku dźwięku ludzkiego głosu) i dla każdego fragmentu w nagraniu sprawdzić jak bardzo ten fragment pasuje do szablonu. Zastosujemy funkcję filtrującą następującej postaci:

$f[n] = [3, 3, 4, 4, -1, -1, -2, -2, -2, -2, -2, -2]$

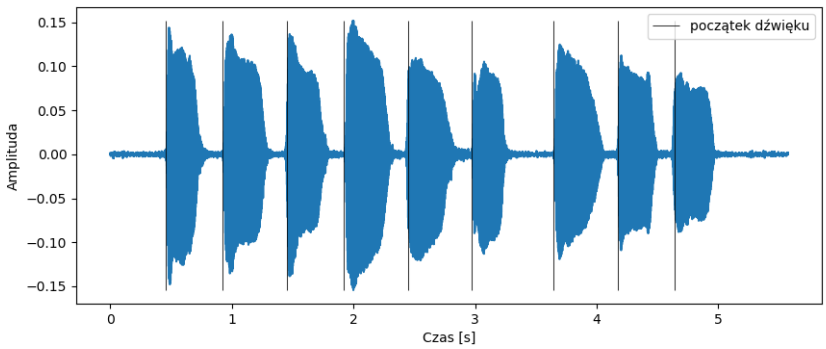
Ostatecznie kroki, które wykonujemy w Envelope match filter wyglądają następująco:

1. $A_k = \max(x[n] | kn_0 \leq n \leq (k+1)n_0)$
   gdzie x to nagranie, które dostajemy na wejściu, a n_0 jest rozmiarem okna
2. $B_k = (\frac{A_k}{0.2+0.1*A_k})^{0.7}$
3. $C_k = \sum_{n=0}^{11} B_{k-n} * f[n]$
   gdzie f jest funkcją dopasowującą
4. If $C_k >$ threshold, to w miejscu kn_0 występuje początek dźwięku

2.4.2. Analiza wyników

Do programu zostało dodane usprawnienie wykorzystujące *epsilon* oraz dodatkowo dodano stałą *gap*, która określa co ile powinniśmy przesuwać okno w każdym

kroku. Zatem możemy mieć na przykład okno $n_0 = 5000$ oraz $gap = 2500$ wtedy w każdym kolejnym kroku przesunięte okno będzie w połowie pokrywało się z poprzednim oknem. Pozwala to dobierać większe okna i jednocześnie wciąż przesuwać się o małe wielkości, co może dawać dokładniejsze wyniki. Zaimplementowany algorytm uruchomiony dla pięciosekundowego nucenia „Wlazł kotek na płotek" z oknem równym 700, progiem 0.5, gap 400 oraz epsilon 3000 daje bardzo dobry wynik (Rysunek 2.16).



Rysunek 2.16: „Wlazł kotek na płotek v.1" Envelope match filter $n_0 = 700, threshold = 0.5, gap = 400, epsilon = 3000$

2.5. Początki dźwięków w dalszych krokach

Do kolejnych kroków potrzebujemy bardzo dokładnych wyników. Niestety wyżej przedstawione algorytmy nie gwarantują idealnego zaznaczenia początków dźwięków. Aby w dalszej części pracy móc analizować poprawność tylko sprawdzanych kroków skorzystamy z modelu „human in the loop". Do dalszej analizy wykorzystamy więc ręczne zaznaczanie onsetów w programie i to te dane przekażemy do kolejnych funkcji [2].

Rozdział 3.

Znajdowanie wysokości dźwięku

Mamy już zaznaczone początki dźwięków w nagraniu. Kolejnym krokiem będzie opisanie tych dźwięków w sposób, który pozwoli na ich późniejszą identyfikację i porównanie z innymi nagraniami. Każdemu dźwiękowi zaczynającemu się w pewnym wcześniej przez nas znalezionym początku odpowiada jakaś nuta. Każdej nucie odpowiada pewna częstotliwość, wysokość. Chcielibyśmy zatem dla każdego z tych dźwięków znaleźć odpowiadającą mu częstotliwość podstawową (inaczej częstotliwość tonu podstawowego). Do analizy poprawności wykorzystamy powszechnie znany, literowy zapis notacji muzycznej: {C, C#, D, D#, E, F, F#, G, G#, A, A#, B}, oznaczając oktawy będziemy trzymać się notacji zachodniej, to znaczy C1, C2, ... C7. W zależności od zapisu europejskiego lub amerykańskiego może pojawiać się również nazwa H zamiast B. Tablica 3.1 przedstawia jakie częstotliwości dźwięku przypisane są konkretnym nutom. Mając częstotliwość możemy wyliczyć nazwę nuty w następujący sposób:

1. $h = 12 * log_2(f/c_0)$
   Gdzie f to częstotliwość szukanego dźwięku a c_0 częstotliwość dźwięku C0 równa 16.35

2. $octave = h//12$

3. $n = h \ modulo \ 12$
   Gdzie n to numer indeksu nuty w tabeli z nazwami: NoteNames = {C, C#, D, D#, E, F, F#, G, G#, A, A#, H}. Końcowym wynikiem będzie wartość $NoteNames[n]$ z dołączonym numerem oktawy $octave$.

Nuta	Częst. (Hz)	Nuta	Częst. (Hz)	Nuta	Częst. (Hz)	Nuta	Częst. (Hz)
C0	16.35	C2	65.41	C4	261.63	C6	1046.50
C#0	17.32	C#2	69.30	C#4	277.18	C#6	1108.73
D0	18.35	D2	73.42	D4	293.66	D6	1174.66
D#0	19.45	D#2	77.78	D#4	311.13	D#6	1244.51
E0	20.60	E2	82.41	E4	329.63	E6	1318.51
F0	21.83	F2	87.31	F4	349.23	F6	1396.91
F#0	23.12	F#2	92.50	F#4	369.99	F#6	1479.98
G0	24.50	G2	98.00	G4	392.00	G6	1567.98
G#0	25.96	G#2	103.83	G#4	415.30	G#6	1661.22
A0	27.50	A2	110.00	A4	440.00	A6	1760.00
A#0	29.14	A#2	116.54	A#4	466.16	A#6	1864.66
B0	30.87	B2	123.47	B4	493.88	B6	1975.53
C1	32.70	C3	130.81	C5	523.25	C7	2093.00
C#1	34.65	C#3	138.59	C#5	554.37	C#7	2217.46
D1	36.71	D3	146.83	D5	587.33	D7	2349.32
D#1	38.89	D#3	155.56	D#5	622.25	D#7	2489.02
E1	41.20	E3	164.81	E5	659.25	E7	2637.02
F1	43.65	F3	174.61	F5	698.46	F7	2793.83
F#1	46.25	F#3	185.00	F#5	739.99	F#7	2959.96
G1	49.00	G3	196.00	G5	783.99	G7	3135.96
G#1	51.91	G#3	207.65	G#5	830.61	G#7	3322.44
A1	55.00	A3	220.00	A5	880.00	A7	3520.00
A#1	58.27	A#3	233.08	A#5	932.33	A#7	3729.31
B1	61.74	B3	246.94	B5	987.77	B7	3951.07

Tablica 3.1: „Wlazł kotek na płotek v.1" – dopasowanie z utworami z serwisu Youtube

Sposoby znajdowania częstotliwości dźwięków można podzielić na te działające w dziedzinie czasu i w dziedzinie częstotliwości. Dziedzinę czasu wykorzystywać będą dwa pierwsze algorytmy: funkcja autokorelacji (Autocorrelation function – ACF) i funkcja różnicy średnich wielkości (Average magnitude difference function – AMDF). Dziedzinę częstotliwości pozostałe.

3.1. Autokorelacja

3.1.1. Przedstawienie algorytmu

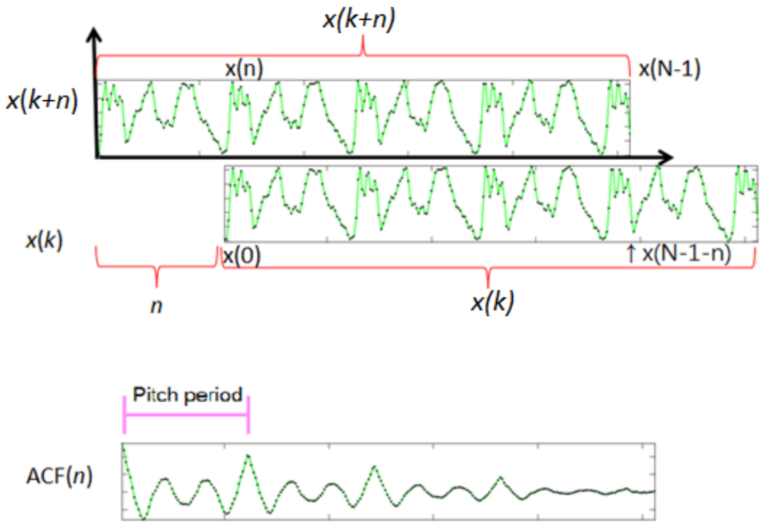
Funkcja autokorelacji lub inaczej autokorelacja (ang. *Autocorrelation function, ACF*) to narzędzie matematyczne często używane w przetwarzaniu sygnałów do analizowania funkcji lub serii wartości. Mniej formalnie, jest to statystyka opisująca w jakim stopniu dany wyraz szeregu zależy od wyrazów poprzednich w szeregu cza-

sowym [1]. Zatem funkcja autokorelacji może pomóc nam znaleźć okres podstawowy P, ponieważ okres ten określa długość najkrótszego fragmentu takiego, że w naszym nagraniu co P elementów występuje bardzo zbliżony pod względem wysokości dźwięku fragment. W rezultacie otrzymamy również częstotliwość podstawową F, ponieważ zachodzi $F = \frac{1}{P}$. Funkcja autokorelacji wygląda następująco:

$$ACF(n) = \frac{1}{N-n} \sum_{k=0}^{N-1-n} x(k)x(k+n)$$

Algorytm ma za zadanie znaleźć w nagraniu wartość n, dla którego funkcja ta przyjmuje największą wartość.

Na rysunku 3.1 znajduje się graficzne przedstawienie tego, co faktycznie robi funkcja ACF. Obliczamy wartość funkcji dla n, które jest przesunięciem w naszym nagraniu x. Funkcja przeiteruje się po każdym możliwym indeksie k takim, że w x istnieje również element o indeksie $k+n$ (czyli od elementu na pozycji 0 do elementu na pozycji $N-1-n$, gdzie N to długość nagrania). Aby zobaczyć jak podobne do siebie są elementy na pozycjach k i $k+n$, mnożymy ich wartości. Dla wszystkich możliwych k bierzemy sumę takich iloczynów, a na końcu dzielimy tę sumę przez liczbę wszystkich jej składników otrzymując ostateczną wartość funkcji.



Rysunek 3.1: Reprezentacja funkcji $ACF(n)$ [13]

3.1.2. Analiza wyników

Jeśli ACF jest największe dla pewnego n, to okres podstawowy wyrażony w sekundach otrzymamy obliczając wartość $K = \frac{n}{sr}$ gdzie sr jest liczbą próbek na sekundę (wartość ta jest uzyskiwana przy wczytywaniu utworu w programie). Niestety złożoność znajdowania wartości dla której ACF jest największe jest kwadratowa i czas działania dla dłuższego nagrania jest duży. Przyjmiemy więc, że szukamy częstotliwości podstawowych z zakresu najczęściej pojawiających się w standardowym

ludzkim śpiewie, czyli od dźwięku G3 do C6, co daje częstotliwości $f_{min} = 200$ Hz i $f_{max} = 1000$ Hz. Wtedy minimalna wartość n, którą chcemy rozważyć wynosi $n = sr/f_{max}$, a maksymalna $n = sr/f_{min}$. Dla standardowej liczby klatek w bibliotece Librosa równej 22050 mamy $n \in < 22, 110 >$. Znacznie zmniejsza to liczbę wywołań pętli przyspieszając działanie programu.

Zgodnie z [10] początkowe dźwięki we „Wlazł kotek na płotek" to: G E E F D D C E G. Nuty, które zwrócił algorytm dla nucenia melodii tej piosenki są następujące: F#4, D#4, D#4, F4, C#4, C#4, B3, D#4, G4. Widzimy, że tylko dwie nuty zostały podane prawidłowo. Sugerowałoby to słaby wynik. Czasami jednak ludzie śpiewają wyżej lub niżej daną melodię, natomiast jeżeli różnice wysokości między kolejnymi dźwiękami są zachowane, to w dalszym wciągu da się wywnioskować z nucenia jaka miała być docelowa piosenka. Przesunięte oryginalne nuty na podstawie zwróconych, które zachowałyby harmonię i melodię oryginału: F#4, D#4, D#4, E4, C#4, C#4, B3, D#4, F#4. Gdy odegramy tę sekwencję na przykład na wirtualnym pianinie [8], to można sprawdzić, że faktycznie brzmi ona prawie jak oryginalna melodia.

W poniższej tabeli *Teoretyczne* oznaczają oryginalne dźwięki piosenki „Wlazł kotek na płotek", *Algorytm ACF* to nuty, które zwrócił program, a *Przesunięte* to oryginalne nuty przesunięte o tyle samo, ile dzieli pierwszą z oryginalnych nut i pierwszą zwróconą przez algorytm (w danej oktawie, czyli w tym przykładzie w czwartej).

Teoretyczne	G E E F D D C E G
Algorytm ACF	F#4, D#4, D#4, F4, C#4, C#4, B3, D#4, G4
Przesunięte	F#4, D#4, D#4, E4, C#4, C#4, B3, D#4, F#4

Nuty zwrócone przez algorytm zgadzają się z przesuniętymi nutami na siedmiu z dziewięciu miejsc co sprawia, że odgrywając je na pianinie, melodia przypomina piosenkę „Wlazł kotek na płotek".

3.2. Funkcja różnicy średnich wielkości

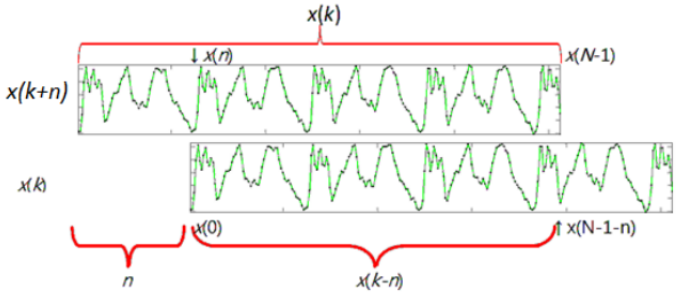
3.2.1. Przedstawienie algorytmu

Funkcja różnicy średnich wielkości (ang. *Average magnitude difference function, AMDF*) jest bardzo podobna do funkcji autokorelacji, wzór jej obliczania przedstawiamy poniżej:

$$AMDF(n) = \frac{1}{(N-n)} \sum_{k=0}^{N-1-n} |x(k) - x(k+n)|$$

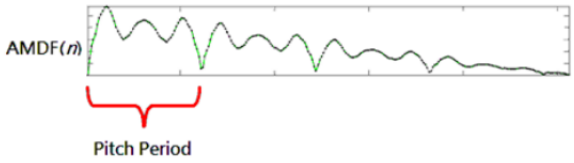
N jest długością analizowanego dźwięku w sygnale x, a n wartością opóźnienia czasowego. AMDF wylicza wielkość różnicy nachodzących na siebie części sygnału – miejsce w którym różnica jest bardzo mała (bliska zeru) możemy uznać za koniec okresu. Innymi słowy szukamy najmniejszego takiego n, że $AMDF(n) \approx 0$, wtedy

n jest okresem dźwięku.



Rysunek 3.2: Dopasowywanie okna przesunięcia [13]

Na rysunku 3.2 widzimy dwa razy ten sam sygnał, w dolnym wierszu przesunięty o n. Funkcja AMDF wylicza sumę różnic między wierszami. Jej wyniki dla kolejnych n zaprezentowano na rysunku 3.3.



Rysunek 3.3: Reprezentacja funkcji $AMDF(n)$ [13]

3.2.2. Analiza wyników

Aby zmniejszyć liczbę wywołań funkcji zastosujemy ograniczenie z poprzedniego podrozdziału, tj. $n \in < 22, 110 >$. Podobnie jak przy metodzie ACF, program został uruchomiony na nagraniu nucenia „Wlazł kotek na płotek". Nuty zwrócone przez algorytm: F#4, F#4, D#4, E4, C#4, C#4, C#4, C#4, F#4. Widać, że w żadnej z nut algorytm nie pokrywa się z ogólnie znanym zapisem. Sugerowałoby to całkowicie błędne wyniki obliczeń, jednak odpowiednio przesuwając kolejne oryginalne nuty otrzymujemy następujące dane:

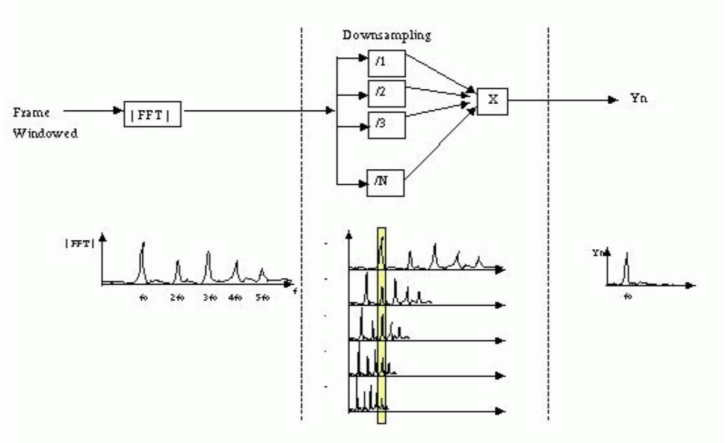
Teoretyczne	G E E F D D C E G
Algorytm AMDF	F#4, F#4, D#4, E4, C#4, C#4, C#4, C#4, F#4
Przesunięte	F#4, D#4, D#4, E4, C#4, C#4, B3, D#4, F#4

Pozwala to stwierdzić, że algorytm zwraca dosyć poprawne wyniki myląc się tylko w trzech dźwiękach.

3.3. Harmonic product spectrum

3.3.1. Przedstawienie algorytmu

W przeciwieństwie do wyżej opisanych metod, Harmonic product spectrum zamiast operować bezpośrednio na nagraniu, oblicza szybką transformację Fouriera (ang. *Fast Fourier Transform, FFT*) i potem to na niej wykonuje obliczenia. Zatem pierwszym krokiem, który wykonujemy, jest policzenie FFT na fragmencie nagrania dla którego chcemy znaleźć częstotliwość tonu podstawowego. Gdy mamy FFT, to dla pewnego wybranego przez nas M bierzemy M próbkowań. Próbkowanie polega na wzięciu tylko niektórych wartości z FFT. Zatem dla każdego $1 \leq m \leq$ M, w m-tym próbkowaniu bierzemy co m-ty element z wcześniej policzonego FFT. Następnie dla każdego indeksu i takiego, że $1 \leq i \leq$ M liczymy sumę wartości na indeksach i w każdym z próbkowań. Jeśli w pewnym próbkowaniu nie występuje i-ty element, to nic nie dodajemy i przechodzimy do kolejnego próbkowania. Dla tak policzonych wartości szukamy największej z nich. Jeśli największa z wartości występuje na pozycji f, to mówimy, że f jest częstotliwością podstawową. Na Rysunku 3.4 znajduje się schemat kolejnych kroków.



Rysunek 3.4: Schemat działania algorytmu HPS [13]

Czyli możemy algorytm zapisać w następujący sposób:

1. $X = FFT(x)$, gdzie x to nagranie
2. $X_m = \mathit{próbkuj}(X, m)$, dla każdego $1 \leq m \leq M$
   gdzie $\mathit{próbkuj}(X, m)$ oznacza wzięcie podzbioru X zawierającego co m-ty jego element.
3. $Y = \sum_{i=1}^{M} X_m$
4. Jeśli w Y największa wartość występuje w miejscu f, to f jest częstotliwością podstawową

3.3.2. Analiza wyników

Inne źródła jak np. [14] mówią, że w trzecim kroku algorytmu HPS zamiast brać sumę można wziąć iloczyn, dlatego w naszej implementacji algorytmu znajduje się iloczyn wartości X_m, a za wartość M zostało przyjęte 4. Poniżej znajduje się wynik programu dla nagrania nucenia „Wlazł kotek na płotek".

Teoretyczne	G E E F D D C E G
Algorytm HPS	F#4, D#4, D#4, E4, C#4, C#4, B3, D#4, F#4
Przesunięte	F#4, D#4, D#4, E4, C#4, C#4, B3, D#4, F#4

Widzimy, że z ogólnie znanym zapisem nie pokrywa się ani jedna zwrócona nuta, jednak uwzględniając przesunięcie wynik zgadza się na każdej nucie.

3.4. Trzecia największa wartość FFT

3.4.1. Przedstawienie algorytmu

Ponieważ nasze wejściowe nucenie i śpiew są wykonywane zawsze przez jedną osobę na raz, nagrania mają tylko jeden ton. Dzięki temu do wykrywania częstotliwości podstawowej możemy użyć metody prostszej niż powyższe. Opierając się na [13] wiemy, że moc składowych harmonicznych jest największa, dzięki temu możemy znaleźć trzy najsilniejsze wartości w dziedzinie częstotliwości i wybrać z nich najniższą. Do tego celu wykorzystamy obliczenia szybkiej transformaty Fouriera.

1. $A = FFT(x)$

   Gdzie FFT jest Szybką Trasformatą Fouriera a x jest sygnałem w dziedzinie czasu.

2. $Frequencies = FFTfreq(len, sr)$

   Gdzie $FFTfreq$ jest tablicą częstotliwości próbkowania FFT dla długości pojedynczego dźwięku i zadanej liczby klatek na sekundę.

3. Weź 3 najwyższe wartości A – f_1, f_2, f_3

4. Jeśli $j = Min(f_1, f_2, f_3)$ to $B(j)$ jest naszą szukaną częstotliwością.

3.4.2. Analiza wyników

Sprawdźmy działanie algorytmu dla nagrania nucenia „Wlazł kotek na płotek".

Teoretyczne	G E E F D D C E G
Algorytm Tnw FFT	F#4, D#4, D#4, E4, C#4, D4, C4, E4, F#4
Przesunięte	F#4, D#4, D#4, E4, C#4, C#4, B3, D#4, F#4

Tak samo jak w poprzednich algorytmach w niewielu nutach algorytm pokrywa się z ogólnie znanym zapisem. Jednak wyliczone dźwięki w dużym stopniu pasują do nut przesuniętych (z wyjątkiem końcowych 3, z których dwie mogłyby być uznane za zaśpiewane czysto).

3.5. Cepstral method

Nazwa metody cepstralnej, pochodzi od anagramu angielskiego słowa spectrum i oznacza spektrum ze spektrum [6]. Pierwotny sygnał dźwiękowy jest w niej przetwarzany za pomocą szybkiej transformaty Fouriera, a wynikowe spektrum jest później konwertowane do skali logarytmicznej. Na wynik drugi raz nakładamy FFT i w rezultacie otrzymujemy cepstrum w dziedzinie czasu, w którym maksima odpowiadają częstotliwościom z pierwotnego sygnału. Najwyższą wartość w cepstrum można uznać za częstotliwość podstawową dźwięku. Przedstawienie działania algorytmu na pojedynczym dźwięku znajduje się na rysunku 3.5. Aby zawęzić wybór i zmniejszyć liczbę błędnych wyników ograniczymy możliwe częstotliwości ramami dla najczęściej nuconych dźwięków $f_{min} = 200$ Hz i $f_{max} = 600$ Hz.

Zapis działania algorytmu:

1. $A = FFT(x)$

   Gdzie FFT jest szybką trasformatą Fouriera a x jest sygnałem w dziedzinie czasu.

2. $Ceps = FFT(ln(x))$

3. $Frequencies = FFTfreq(len, sr)$

   Gdzie $FFTfreq$ jest tablicą częstotliwości próbkowania FFT dla długości liczonego dźwięku i zadanej liczby klatek na sekundę.

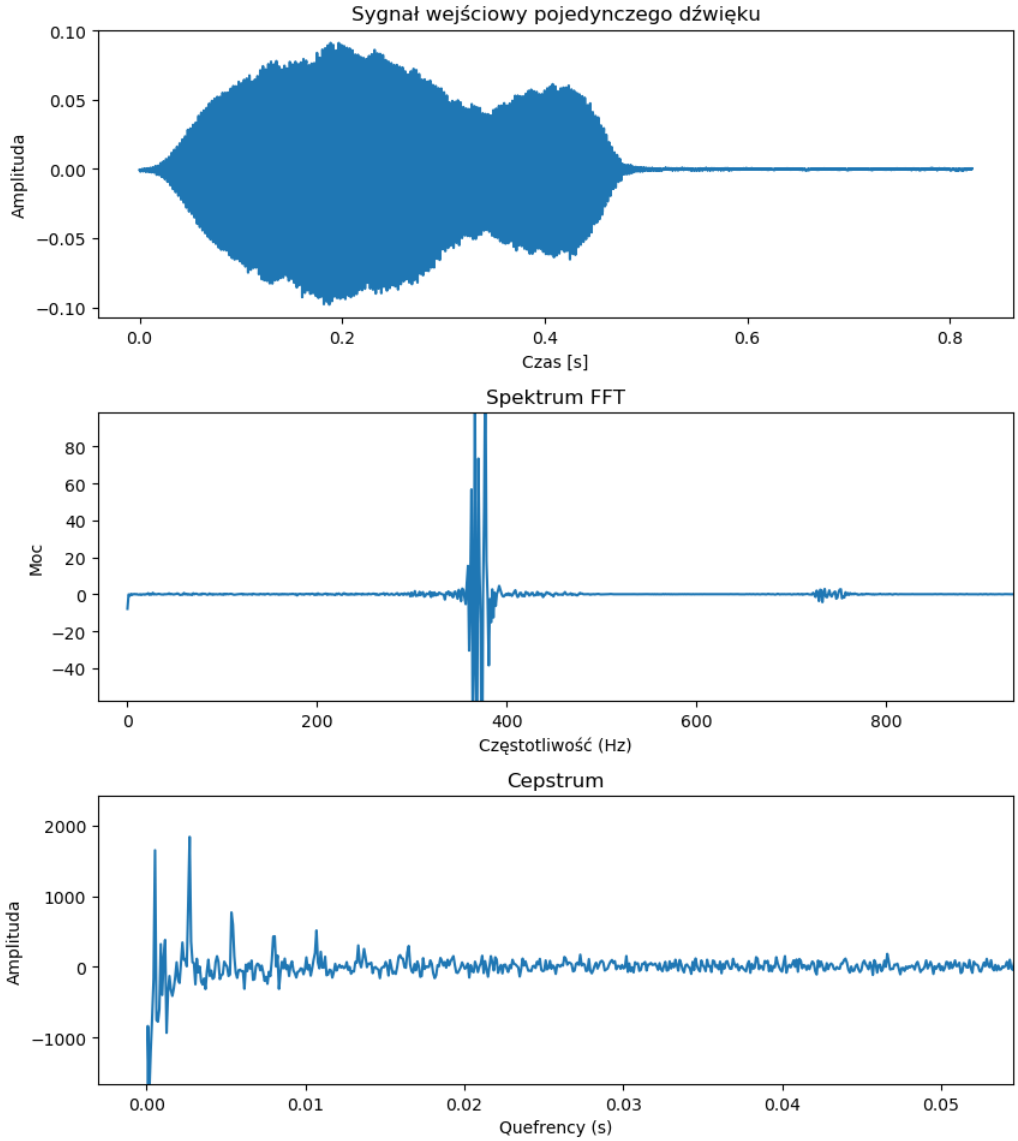
4. $Quefrencies = FFTfreq(len, df)$

   Gdzie df to odległość pomiędzy częstotliwościami we $Frequencies$ (można je obliczyć sprawdzając różnicę między $Frequencies[0]$ a $Frequencies[1]$).

5. $valid = (Quefrencies > \frac{1}{f_{max}})\&(Quefrencies \leq \frac{1}{f_{min}})$

   Wektor $valid$ opisuje, które indeksy z $Quefrencies$ spełniają warunek mieszczenia się w ramach częstotliwości a & jest tutaj operacją bitową.

6. Jeśli j to największy poprawny element w $Ceps$ wtedy $\frac{1}{Quefrencies[valid][maxIndx]}$ jest szukaną częstotliwością podstawową.



Rysunek 3.5: Reprezentacja funkcji $Ceps(n)$

Na znajdowanie największego elementu w *Ceps* można spojrzeć z dwóch stron. Jedno rozwiązanie po prostu zwraca maksymalną wartość (I), drugie uwzględnia tylko te elementy, które są lokalnymi maksimami funkcji (II). Dzięki temu możemy ograniczyć pojawianie się wartości brzegowych, chyba że faktycznie pojawiło się tam maksimum.

3.5.1. Analiza wyników

Do analizy wykorzystamy analizowane już wcześniej nucenie „Wlazł kotek na płotek".

Teoretyczne	G E E F D D C E G
Algorytm I	F#4, D#4, D4, E4, C4, C#4, B3, D#4, F#4
Algorytm II	F#4, D#4, D#4, E4, C#4, C#4, B3, E4, F#4
Przesunięte	F#4, D#4, D#4, E4, C#4, C#4, B3, D#4, F#4

Algorytm I to nuty zwrócone przez algorytm wykorzystujący literalne wybieranie największego elementu. *Algorytm II* to nuty zwrócone przez algorytm wykorzystujące wybieranie największego elementu wśród maksimów.

Wyliczone dźwięki obu wersji w bardzo dużym stopniu pasują do nut przesuniętych. Jedyne pojawiające się pomyłki to błędy rzędu jednej nuty. Wersja II ma tylko jeden błąd. Sugeruje to dużą poprawność obliczeń.

3.6. Częstotliwości w dalszych krokach

Metoda cepstralna I, Metoda cepstralna II, Trzecia największa wartość FFT i Harmonic product spectrum wszystkie zwracają rozsądne rezultaty dla naszych testowanych obliczeń. Powyższe cztery algorytmy mylą się w różnych miejscach, dlatego rozsądnym podejściem będzie analizowanie wszystkich wyników i na ich podstawie wyliczanie końcowej listy nut. Dodatkowo, aby ograniczyć problem losowości wyboru w przypadku nierozstrzygalnych wyników wybierzemy algorytm Harmonic product spectrum jako priorytetowy. Działanie algorytmu wybierającego nuty:

1. Oznaczmy jako *Preferowane* rozwiązanie priorytetowe, a jako *Nuty* wynikową tablicę nazw nut.
2. Kolejne dźwięki analizujemy osobno. $N(x)$ to zliczona ilość wystąpień dźwięku x na i-tym miejscu we wszystkich tablicach. Mogą być co najwyżej 4 różne wartości x. Niech Max będzie największą wartością w N a y odpowiadającą jej nutą. W zależności od liczby wystąpień nuty y:
   (a) Jeżeli $Max \geq 3$ to dopisz y do $Nuty$
   (b) Jeżeli $Max = 2$ i istnieje takie z, że $y \neq z$ i $N(z) = 2$ to dopisz $Preferowane[i]$ do $Nuty$
   (c) Jeżeli $Max = 2$ i nie istnieje takie z, że $y \neq z$ i $N(z) = 2$ to dopisz y do $Nuty$
   (d) W przeciwnym przypadkach zapisz $Preferowane[i]$ do $Nuty$.
3. Dla każdego elementu w $Nuty$ zapisz w wynikowej tablicy f odpowiadającą mu idealną częstotliwość.

Nuty przeliczamy na częstotliwości w następujący sposób [5]:

1. Znajdź indeks i szukanej nuty w tablicy {C, C#, D, D#, E, F, F#, G, G#, A, A#, H}
2. $v = i + 4 + ((oktawa - 1) * 12)$
3. Końcowa wartość to $(A4 * 2^{(v-49)/12})$ zaokrąglone do drugiego miejsca po przecinku, a $A4$ to wysokość dźwięku $A4$ równa 440 Hz.

Dzięki zapisaniu flagowych częstotliwości, a nie ich wahań w kolejnym kroku przy porównaniach z formatem MIDI nie będziemy musieli uwzględniać drobnych różnic.

Rozdział 4.

Dopasowywanie melodii

Gdy mamy znalezione tony podstawowe odpowiadające kolejnym dźwiękom, pozostał do omówienia ostatni etap potrzebny do określenia podobieństwa naszego nagrania z piosenkami z bazy danych. Oba niżej przedstawione algorytmy podają w jakim stopniu nasze nagranie jest zgodne z innym nagraniem. Na początku wcześniej otrzymany ciąg częstotliwości podstawowych przekonwertujemy do ciągu liczb MIDI. Piosenki w bazie również przechowywane są w tym formacie. Jest to spowodowane faktem, że piosenki przedstawione jako ciąg MIDI są łatwo dostępne i wystarczy jednie umieścić je w bazie, nie trzeba dokonywać żadnych dodatkowych konwersji. Dobry algorytm określający podobieństwo dwóch nagrań powinien uwzględniać fakt, że tempo nucenia/śpiewania będącego zapytaniem może być wolniejsze lub szybsze niż oryginalnego utworu oraz wysokości dźwięków również mogą się lekko różnić od tych oryginalnych. W tym rozdziale przedstawimy analizę dwóch algorytmów, które mogą rozwiązywać problem dopasowywania melodii.

Format MIDI

Plik w formacie MIDI, który otrzymujemy zawiera kanał, a w nim są ścieżki zawierające wiadomości. Wiadomości przekazują wiele informacji, nas będą interesować dwa rodzaje – oznajmująca o początku lub końcu brzmienia nuty oraz o tempie wybrzmiewania dźwięków. Przykładowa wiadomość o początku brzmienia nuty może wyglądać następująco:

< message note on channel=0 note=54 velocity=51 time=0 >

Każda wiadomość posiada informację o pewnej nucie. Na pierwszej pozycji możemy mieć zmienną *„note on”* lub *„note off”*, które odpowiednio oznaczają, że dana nuta pojawiła się w tym czasie lub zniknęła. Interesuje nas jeszcze pole *note* mówiące jaka to nuta (w notacji MIDI, nie w zapisie dźwiękowym, którym posługiwaliśmy się dotychczas) oraz *time* (w formacie *ticks per beat*), podający deltę czasu, czyli ile „tyknięć” ma minąć od poprzedniej wiadomości. Na podstawie *time* możemy

obliczyć czas w sekundach dla danej wiadomości.

Otrzymanie bezwzględnego czasu wystąpienia nuty

Na początku pliku MIDI podane jest tempo utworu, albo raczej jest ono ustawiane poleceniem *set_tempo* zawartym w wiadomości. Potrzebujemy też informacji o wartości *ticks_per_beat* zawartej w nagłówku pliku. Aby otrzymać bezwzględny czas należy przejść po wszystkich wiadomościach w ścieżce i zliczać liczbę tyknięć dla każdej z nich sumując kolejne czasy. Wtedy do przeliczenia czasu wykorzystamy wbudowaną w bibliotekę mido funkcję *tick2second()* przyjmującą wartości bezwzględnego czasu, *ticks_per_beat* oraz tempa. Działanie funkcji *tick2second()* można opisać w następujący sposób [4][9]:

1. $sPerTick = \frac{tempo}{ticks\_per\_beat}$
2. $secondsPerTick = \frac{sPerTick}{1.000.000}$
3. $seconds = ticks * secondsPerTick$

   Gdzie *ticks* to bezwzględny czas wystąpienia nuty

4.1. Programowanie dynamiczne

Jest to dość prosty do zaimplementowania algorytm, którego idea również nie jest skomplikowana. Tak jak wyżej zostało wspomniane, porównujemy ze sobą dwa ciągi liczb MIDI. Niech Q będzie ciągiem odpowiadającym zapytaniu, P ciągiem z bazy danych, a $|Q|$ i $|P|$ odpowiadającym im długościom. Niech q będzie prefiksem ciągu Q o długości i, a p prefiksem ciągu P o długości j, $1 \leq i \leq |Q|$, $1 \leq j \leq |P|$. Algorytm polega na policzeniu macierzy o rozmiarze $(|Q| + 1) \times (|P| + 1)$. W komórce (i, j) przechowujemy informację jaki jest największy możliwy wynik „wyrównania" biorąc prefiks q i p. Wynik wyrównania mówi nam na ile podobne są dwa ciągi, gdy dodamy do nich puste pola w dowolnych miejscach tak, żeby oba były tej samej długości. Korzystamy z wyników dla krótszych prefiksów, które zostały policzone wcześniej. Jeśli dwa ostatnie elementy q i p są takie same, to możliwy wynik wyrównania to wynik q i p krótszych o jeden znak zwiększony o 2. Jeśli te dwa elementy są różne, to możliwy wynik, to wynik q i p krótszych o jeden znak, ale musimy odjąć od niego 2, ponieważ chcemy zawrzeć informację, że przy tym wyrównaniu mamy dodatkową niepasującą parę. Musimy też rozważyć wyrównania, które wstawiają pusty znak na koniec q lub p. Dodanie pustego znaku powoduje, że od wyniku trzeba odjąć 1, zatem inne możliwe wyniki to wynik wyrównania z komórki $(i - 1, j)$ pomniejszony o 1 oraz wynik wyrównania z komórki $(i, j - 1)$ pomniejszony o 1. Dla prefiksów q i p chcemy wybrać największy z wszystkich wyżej wymienionych możliwych wyników. Aby dalsze obliczenia przebiegały poprawnie i

nie trzeba było sprawdzać wykraczania poza macierz dodatkowo tworzymy kolumnę o indeksie 0 oraz wiersz o indeksie 0 i uzupełniamy komórki $(i, 0)$ wartościami $-i$, a na pozycji $(0, j)$ wartościami $-j$. Na pozycji $(0, 0)$ jest wartość 0. Poniżej znajduje się zwięźle napisana definicja:

$$wynikWyrównania(i,j) = \begin{cases} wynikWyrównania\text{(i-1, j-1)} + wynikDopasowania(q[i], p[j]) \\ wynikWyrównania\text{(i-1, j)} - 1 \\ wynikWyrównania\text{(i, j-1)} - 1 \end{cases}$$

gdzie

$$wynikDopasowania(a,\ b) = \begin{cases} 2, & jeśli\ a = b. \\ -2, & w\ przeciwnym\ przypadku. \end{cases}$$

4.1.1. Analiza wyników

Ciąg P, będący utworem z którym porównujemy nasze zapytanie, wczytujemy w Pythonie za pomocą biblioteki mido. Aby uwzględnić pomyłkę o jedną nutę niżej/wyżej w nuconym zapytaniu, *wynikDopasowania(a,b)* sprawdzał czy $a \in [b-1, b+1]$. W poniższej tabeli przedstawione są wyniki dopasowań nucenia „Wlazł kotek na płotek v.1" z czterema innymi nuceniami. Nucenia te nie mają żadnych szumów w tle, zawierają jedynie konkretną melodię:

Nazwa porównywanego nucenia	Liczba nut	Wynik
WlazłKotek v.1	28	11
WlazłKotek v.2	19	10
KawalekPodlogi v.2	24	2
SzlaDzieweczka v.1	113	3

Tablica 4.1: „Wlazł kotek na płotek v.1" – dopasowanie z utworami nuconymi

Widać, że najwyższe wyniki są faktycznie dla nagrań z zanuconym "Wlazł kotek". Zauważmy, że gdy ciągi Q i P mają dużą różnicę w długościach, to żeby znaleźć wynik dla ich wyrównania, trzeba wstawić dużo pustych pól. Wiąże się to z dużą „karą" za wstawianie pustych pól, którą jest odejmowanie 1 od wcześniej wyliczonego wyniku (2 i 3 przypadek w definicji *wynikWyrównania*). „Wlazł kotek v.1" zawiera więcej nut niż „Wlazł kotek v.2", stąd prawdopodobnie niższy wynik.

Następnie zostały sprawdzone wyniki porównania „Wlazł kotek v.1" z dwoma wersjami piosenki „Wlazł kotek na płotek" pobranymi z serwisu Youtube[1]. Jedna z nich to utwór śpiewany zawierający podkład muzyczny, a druga to wersja zagrana na gitarze.

[1] `https://www.youtube.com/`

Nazwa porównywanego nucenia	Liczba nut	Wynik
WlazłKotek1	375	4
WlazłKotek2	85	2

Tablica 4.2: „Wlazł kotek na płotek v.1" – dopasowanie z utworami z serwisu Youtube

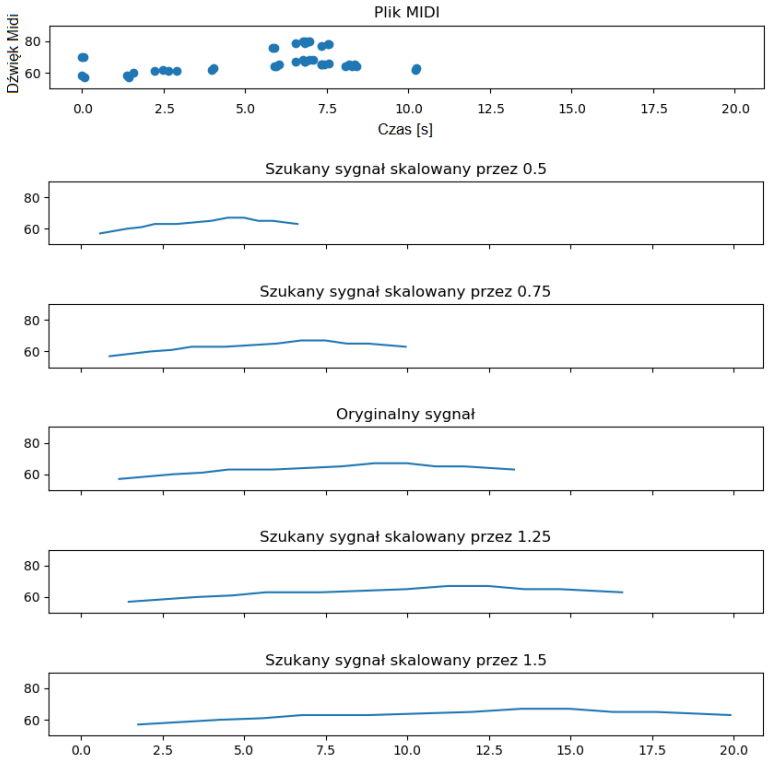
Mimo, że wersje z Youtube również przedstawiały melodię „Wlazł kotek na płotek", to program poradził sobie gorzej niż z dopasowywaniem do wersji nuconych. Zwróćmy uwagę, że WlazłKotek1 w tablicy 4.2 ma bardzo dużo nut. Mimo, że melodie z porównywanych nagrań mogą być do siebie podobne, to w tej reprezentacji ciągów nut chcąc wyrównać je długością będziemy musieli wstawić sporo pustych pól. Zauważmy również, że algorytm nie uwzględnia faktu, że ktoś może śpiewać oktawę niżej/wyżej niż nagranie z którym dokonywane jest porównanie. Może to powodować zawyżone wyniki dopasowań z utworami, które nie są szukaną piosenką, ale również zaniżone wyniki dla nagrań reprezentujących faktycznie szukany utwór, a tego chcielibyśmy uniknąć.

4.2. Skalowanie liniowe

Algorytm skalowania liniowego jest matematycznym odwzorowaniem ludzkiego dopasowywania do siebie dźwięków. Opiera się na dopasowywaniu dźwięków na poziomie kolejnych klatek i z tego powodu dane muszą nie tylko zawierać informację o wysokości i kolejności dźwięków ale też czasie wystąpienia – potrzeba danych o rytmie zapisanej melodii. Nucąc piosenkę człowiek nie jest w stanie zaśpiewać kolejnego dźwięku w idealnie tym samym czasie jak w oryginale. Czasem zaśpiewa się szybciej, czasem wolniej. Z reguły te niepoprawności mieszczą się w granicach 0.5 do 1.5 raza prędkości piosenki. I właśnie te odchylenia pozwala nam przeanalizować algorytm skalowania liniowego (rysunek 4.1). Z oryginalną piosenką porównujemy nagranie przeskalowane przez współczynniki między 0.5 a 1.5.

Jako, że w konkretnym momencie może występować więcej niż jedna nuta, to analizując sygnał z bazy danych (nazywany dalej *targetMIDI*) tworzymy kubełki które zawierają nuty występujące w tym samym czasie. Są to wiadomości mające za *time* wpisaną wartość 0 – czyli od poprzedniej wiadomości do tej nie minął żaden czas. Gdy pojawia się nowa nuta, bierzemy poprzedni kubełek i dopisujemy do niego tę nutę. Jeśli jakaś nuta znika, to kolejny kubełek jest kopią poprzedniego oprócz tej nuty. W ten sposób tworzymy ciąg kubełków zawierających nuty występujące w tym samym momencie. Jednocześnie analizując kolejne dźwięki naszego nagrania (nazywanego *queryMIDI*) musimy odwołać się do znalezionych w pierwszym rozdziale początków nut i na podstawie numeru elementu w którym wystąpił onset wyliczyć czas jego wystąpienia ze wzoru $czas = \frac{i}{sr}$ gdzie i jest numerem indeksu początku a sr to, jak wcześniej, liczba klatek na sekundę. Zamieniamy też częstotliwości idealne,

wyliczone w poprzednim rozdziale na wartości MIDI. Mając tak przygotowane dane możemy przejść do ich analizy.



Rysunek 4.1: Reprezentacja działania funkcji skalowania liniowego

Działanie funkcji skalowania liniowego najelepiej będzie pokazać na fragmencie kodu (Rysunek 4.2).

```
86      for factor in scaling:
87          for noteList,time in targetMIDI:
88      [1]     if timesInput[queryIter]*factor+0.08 < time:
89                  if queryIter< len(timesInput)-1: queryIter += 1
90      [2]     if time >= timesInput[queryIter]*factor-0.08 \
91                  and time <= timesInput[queryIter]*factor+0.08:
92          [2.1]   if (queryMIDI[queryIter] in noteList) \
93                      or (queryMIDI[queryIter]+1 in noteList) \
94                      or (queryMIDI[queryIter]-1 in noteList):
95                      matchingFactor[i] += 1
96                      queryIter += 1
97          [2.2]   elif (queryMIDI[queryIter] in previous):
98                      matchingFactor[i] += 1
99                      queryIter += 1
100             previous = noteList
101         i += 1
102         queryIter = 0
103
```

Rysunek 4.2: Fragment kodu funkcji skalowania liniowego

Na początku zdefiniujmy, że *scaling* = {0.5, 0.75, 1, 1.25, 1.5}, a *timesInput*

to tablica czasów odpowiadających onsetom w *queryMIDI*. Zgodnie z opisem algorytmu musimy sprawdzać nasze dźwięki z *targetMIDI* i właśnie to robimy w miejscu kodu oznaczonym [2] (linijka 90 w kodzie na Rysunku 4.2)). Oczekiwanie, że dźwięki w dwóch różnych nagraniach pojawią się w dokładnie tej samej chwili jest naiwne, ustawimy więc przedział czasu w jakim mogą się pokryć. Następnie, jeżeli czasy się zgadzają to sprawdzimy czy szukana nuta znajduje się w nutach z targetu. Jak sprawdziliśmy w poprzednim rozdziale ludzie śpiewają czasem niepoprawnie – myląc się w wysokości dźwięku, chcemy więc też dopuścić możliwość pomyłki o jeden dźwięk w górę lub w dół. Jeżeli w którejś z tych wersji znajdziemy dopasowanie to zwiększamy *matchingFactor*[i] o jeden. Tablica *matchingFactor* przechowuje wyniki, które są sumą dopasowań dla kolejnych współczynników, gdzie i jest indeksem współczynnika. Uwzględniając kubełkową budowę *targetMIDI* w naszym algorytmie dopisaliśmy jeszcze sprawdzenie (oznaczone w kodzie [2.2]), czy może szukany przez nas dźwięk dopiero co się nie skończył, ponieważ w takim przypadku i tak chcemy zwiększyć wartość sumy dopasowań. Z powodu, że nie wszystkie elementy z *queryMIDI* uda się dopasować do *targetMIDI* to w momencie gdy przekroczymy czas w sygnale z bazy chcemy przejść do kolejnego dźwięku w query (miejsce w kodzie oznaczone [1]).

4.2.1. Analiza wyników

Sprawdźmy wyniki porównania naszego przewodniego utworu „Wlazł kotek v.1".

Utwór	Wynik	Współczynnik
WlazłKotek v.1	3	0.5
WlazłKotek v.2	1	0.5
GdyŚlicznaPanna	0	0.5
SzłaDzieweczka	1	0.75

Tablica 4.3: „Wlazł kotek na płotek v.1" – dopasowanie z utworami nuconymi

Faktycznie, algorytm największe podobieństwo znajduje między poprawnymi utworami. Nucenie jest jednak bardzo krótkie więc różnice są niewielkie. W przypadku dłuższych, bardziej rozbudowanych utworów a zwłaszcza dla nagrań posiadających podkład muzyczny dobrze jest dopuścić możliwość błędnego (nadmiernego) dopasowania piosenek niepoprawnych. Dlatego do dalszej analizy użyjemy algorytmu, który uwzględnia średnią sumę wyników dla jednego tytułu piosenki zamiast rozważać je indywidualnie. Dlatego do kolejnego testu, w którym sprawdzimy już prawdziwe nagrania z internetu, użyjemy przynajmniej dwóch nagrań tych samych tytułów i dopiszemy dodatkową kolumnę oznaczającą średnią arytmetyczną wartości wyników.

Utwór	Wynik	Współczynnik	Średnia
WlazłKotek1	5	1.25	3
WlazłKotek2	1	0.5	
GdySlicznaPanna1	3	0.75	1.5
GdySlicznaPanna2	0	0.5	

Tablica 4.4: „Wlazł kotek na płotek v.1” – dopasowanie z utworami nuconymi

Widzimy, że wyniki dopasowania są poprawne – najwyższa średnia jest dla nagrań przedstawiających oryginały piosenki – pierwsze z nich jest śpiewem z podkładem muzycznym, drugie to dźwięki grane na gitarze. Natomiast pierwsze nagranie „Gdy śliczna panna” śpiewane jest męskim głosem i z głośnym podkładem muzycznym, drugie zostało zagrane na pianinie. Zauważmy jednak, że ten algorytm podobnie jak programowanie dynamiczne również nie uwzględnia, że ktoś może śpiewać oktawę niżej/wyżej niż nagranie z którym dokonywane jest porównanie.

Rozdział 5.

Porównanie algorytmów

Każdy z algorytmów został przetestowany dla 3 piosenek zanuconych przez dwie różne osoby, czyli łącznie dla sześciu nagrań.

Osoby tworzące bazę do testowania miały za zadanie zanucić odpowiednie fragmenty utworów:

- „Gdy śliczna Panna":

  Gdy śliczna Panna Syna kołysała

  z wielkim weselem tak Jemu śpiewała.

- „Szła dzieweczka do laseczka":

  Szła dzieweczka do laseczka

  do zielonego a-haa, do zielonego a-haa, do zielonego.

- „Mój jest ten kawałek podłogi":
  Trąbki rozpoczynające oryginalne wykonanie piosenki.

W poniższych sekcjach omawiamy testy dla znajdowania początków dźwięków, wysokości dźwięków oraz dopasowywania melodii.

5.1. Testy metod znajdujących początki dźwięków

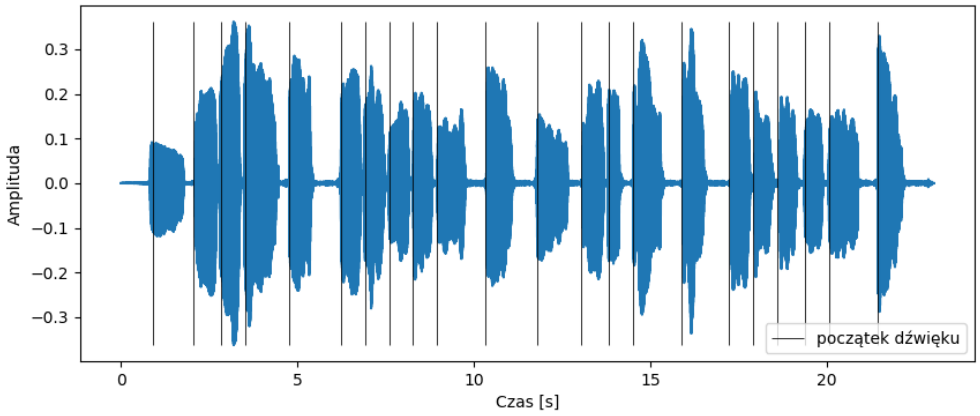
Przy wykrywaniu początków dźwięków wyniki programu zależą od wartości stałych, które są ustawianie ręcznie przed uruchomieniem go. Przy ocenie zwracanych odpowiedzi brane pod uwagę było ile początków zostało zwróconych prawidłowo (w tabeli oznaczone jako *Poprawne*), ile zostało podanych w miejscach, w których nie powinny się znajdować (*Nadmierne*), a także ilu program nie podał, a powinien był (*Nieoznaczone*). Dla każdego z nagrań przedstawiamy wyniki dla przykładowych rozmiarów okna(n_0), *epsilon* oraz progu (*threshold*). Dodatkowo dla

algorytmu Envelope match filter uwzględniona została wartość *gap*. W tabeli stała *threshold* oznaczona została literą t, *epsilon* literą ϵ, a *gap* literą g. Do każdego z testów załączony jest wykres najlepszego wyniku z podpisem przy jakiej metodzie oraz jakich parametrach został on uzyskany.

Gdy śliczna Panna v.1

Nazwa metody	Wartości stałych	Poprawne	Nadmierne	Niezaznaczone
Magnitude method	t=0.02 n_0=1000 ϵ = 6000	22	6	0
	t=0.02 n_0=1000 ϵ = 6000	22	6	0
	t=0.015 n_0=2000 ϵ = 6000	22	2	0
Short-term energy method	t=2 n_0=1000 ϵ = 6500	22	3	0
	t=5 n_0=1000 ϵ = 6500	21	2	1
	t=6 n_0=2500 ϵ = 6500	22	0	0
Surf method	t=0.02 n_0=2000 ϵ = 6000	14	0	1
	t=0.015 n_0=1500 ϵ = 6000	22	0	0
	t=0.02 n_0=1000 ϵ = 6000	22	0	0
Envelope match filter	t=2 n_0=1000 ϵ = 6500 g=400	22	1	0
	t=5 n_0=1000 ϵ = 6500 g=400	22	0	0
	t=5 n_0=500 ϵ = 6500 g=400	22	0	0

Tablica 5.1: „Gdy śliczna Panna v.1”



Rysunek 5.1: Short-term energy method, „Gdy śliczna panna v.1”, $n_0 = 2500$, *threshold* = 6, *epsilon* = 6500

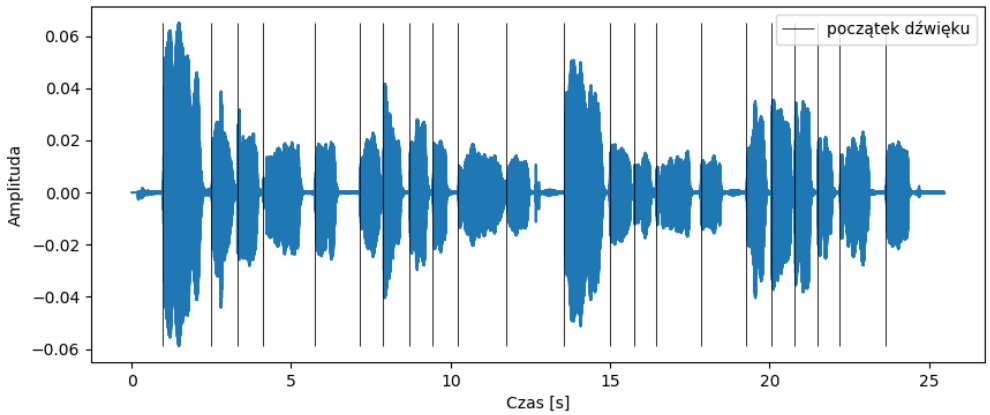
W Short-term energy method kiedy okno wynosiło 1000 i próg był równy 2, program zwracał nadmiarowe początki. Jeden z nich znajdował się na końcu jednego z dźwięków, gdzie amplituda była dość wysoka, a jego odległość od innych początków była większa niż ϵ=6500. Poprzez wydłużenie okna wynik dla tego nadmiarowego

początku brał pod uwagę cichy fragment. W Surf method dla t=0.02 n_0=2000 i ϵ = 6000 spotkaliśmy się z sytuacją, gdzie zaznaczone początki dźwięków były w zbyt dużej odległości od faktycznego miejsca rozpoczęcia – nie uznajemy więc ich ani za nadmierne ani za nieoznaczone. Było 7 takich przypadków, wynikają one ze specyfiki działania przybliżania w algorytmie. Po zmniejszeniu rozmiaru okna o 500 wszystkie odległości zmniejszyły się a po ustawieniu tej zmiennej na 1000 wszystkie onsety były idealnie zaznaczone.

Gdy śliczna Panna v.2

Nazwa metody	Wartości stałych	Poprawne	Nadmierne	Niezaznaczone
Magnitude method	t=0.015 n_0=2000 ϵ = 6000	11	0	11
	t=0.02 n_0=1000 ϵ = 6000	10	0	12
	t=0.01 n_0=1000 ϵ = 6000	14	1	8
	t=0.006 n_0=1000 ϵ = 6000	21	3	1
Short-term energy method	t=0.01 n_0=2500 ϵ=7000	22	4	0
	t=0.05 n_0=2000 ϵ=7000	22	2	0
	t=0.001 n_0=800 ϵ = 6000	22	0	0
Surf method	t=0.01 n_0=1000 ϵ = 6000	4	0	18
	t=0.007 n_0=1000 ϵ = 6000	6	0	16
	t=0.007 n_0=200 ϵ = 6000	6	0	16
Envelope match filter	t=0.2 n_0=1000 ϵ=5500 g=400	19	5	3
	t=0.7 n_0=1800 ϵ=6500 g=600	20	0	2
	t=0.4 n_0=2500 ϵ=6500 g=800	22	0	0

Tablica 5.2: „Gdy śliczna Panna v.2”



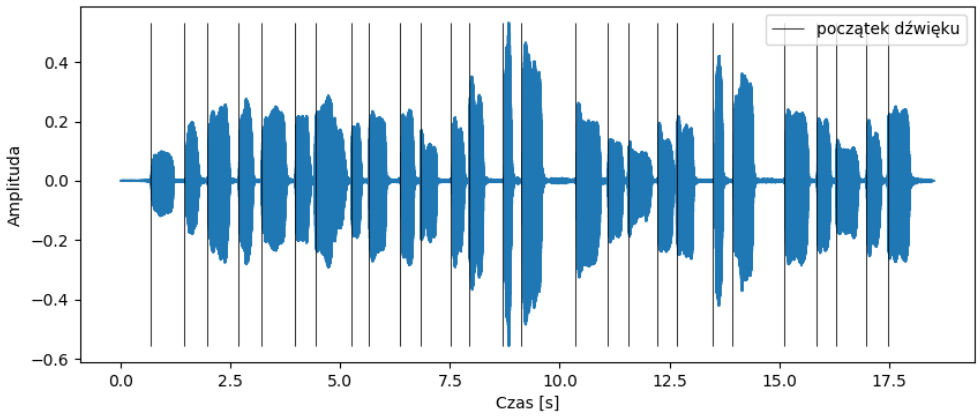
Rysunek 5.2: Short-term energy method, „Gdy śliczna panna v.2”, n_0 = 800, *threshold* = 0.001, *epsilon* = 6000

Analizując tabelę 5.2 widać, że dla nagrań z szybko rosnącą amplitudą głośności dźwięku algorytm Surf method daje bardzo słabe wyniki - ciężko było dobrać stałe.

Szła dzieweczka v.1

Nazwa metody	Wartości stałych	Poprawne	Nadmierne	Niezaznaczone
Magnitude method	t=0.006 n_0=1000 ϵ = 6000	25	1	2
	t=0.01 n_0=1000 ϵ = 6000	26	0	1
	t=0.01 n_0=1000 ϵ = 4000	27	2	0
	t=0.01 n_0=1000 ϵ = 5500	27	0	0
Short-term energy method	t=0.5 n_0=500 ϵ=5500	23	1	4
	t=1 n_0=1000 ϵ=5500	24	1	3
	t=3 n_0=1500 ϵ = 5500	27	0	0
Surf method	t=0.01 n_0=1000 ϵ = 6000	27	0	2
	t=0.01 n_0=1000 ϵ = 4000	27	0	0
	t=0.01 n_0=800 ϵ = 4000	27	1	0
Envelope match filter	t=0.5 n_0=500 ϵ = 5500 g=400	22	0	5
	t=3 n_0=1500 ϵ=5500 g=400	27	0	0
	t=1 n_0=1000 ϵ=5500 g=400	27	0	0

Tablica 5.3: „Szła dzieweczka v.1”



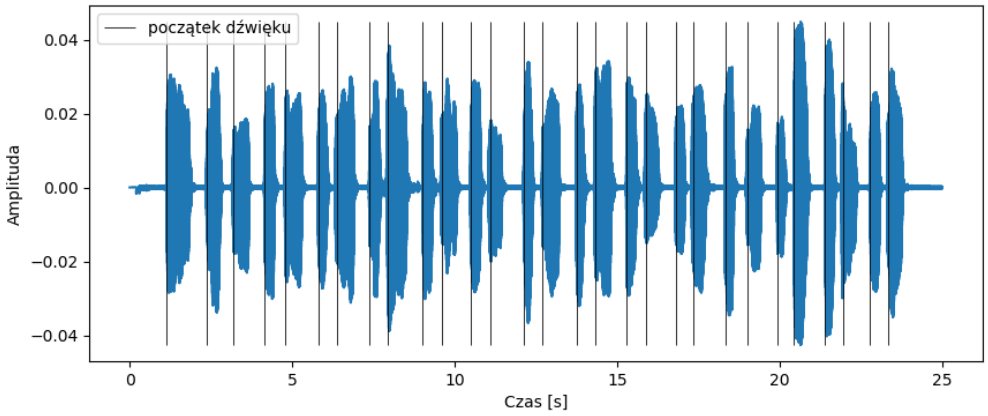
Rysunek 5.3: Envelope match filter, „Szła dzieweczka v.1”, $n_0 = 1500$, $threshold = 3$, $epsilon = 5500$, gap = 400

W Envelope match filter okno wynoszące 500 przy progu równym 0.5 jest za krótkie, ponieważ nie podaje wszystkich wyników. Zwiększenie okna sprawia, że wyniki dla każdego z okien będą wyższe, zatem zwiększyliśmy również próg, co pomogło uzyskać prawidłową odpowiedź.

Szła dzieweczka v.2

Nazwa metody	Wartości stałych	Poprawne	Nadmierne	Niezaznaczone
Magnitude method	t=0.01 n_0=1000 ϵ = 6000	28	0	1
	t=0.006 n_0=1000 ϵ = 6000	29	0	0
	t=0.008 n_0=1000 ϵ = 6000	29	0	0
Short-term energy method	t=0.05 n_0=500 ϵ=6500	20	1	9
	t=0.01 n_0=700 ϵ=5500	29	1	0
	t=0.05 n_0=1500 ϵ = 6000	29	0	0
Surf method	t=0.008 n_0=1000 ϵ = 5000	11	0	18
	t=0.008 n_0=8000 ϵ = 5000	9	0	20
	t=0.003 n_0=500 ϵ = 5000	29	0	0
Envelope match filter	t=0.4 n_0=2700 ϵ=6500 g=600	28	0	1
	t=0.3 n_0=600 ϵ=6500 g=600	29	0	0
	t=0.3 n_0=300 ϵ = 6500 g=600	29	0	0

Tablica 5.4: „Szła dzieweczka v.2”



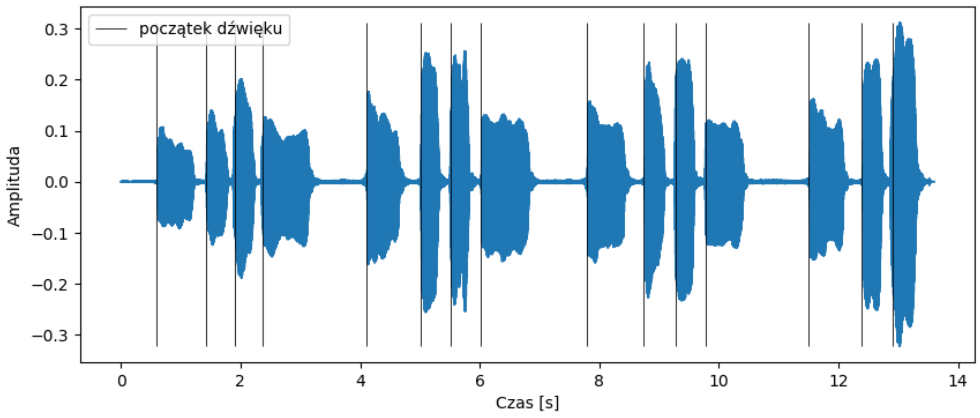
Rysunek 5.4: Envelope match filter, „Szła dzieweczka v.2”, $n_0 = 300$, *threshold* = 0.3, *epsilon* = 6500, gap = 600

W metodzie Envelope match filter już przy pierwszych dobranych wartościach algorytm zadziałał całkiem dobrze, bo tylko jeden początek nie został zaznaczony. Po zmniejszeniu okna z 2700 do 600 i progu z 0.4 do 0.3 wynik był już prawidłowy. Okazało się, że okno można było zmniejszyć jeszcze do 300 i wynik wciąż się zgadzał. Z kolei w wypadku algorytmu Short-term energy method prawidłowy wynik został uzyskany po zwiększeniu rozmiaru okna oraz modyfikacji ϵ.

Mój jest ten kawałek podłogi v.1

Nazwa metody	Wartości stałych	Poprawne	Nadmierne	Niezaznaczone
Magnitude method	t=0.008 n_0=1000 ϵ = 6000	14	2	1
	t=0.01 n_0=1000 ϵ = 6000	14	0	1
	t=0.01 n_0=1000 ϵ = 6000	15	0	0
Short-term energy method	t=3 n_0=1500 ϵ=5500	15	0	0
	t=1 n_0=1000 ϵ=5500	15	2	0
	t=1 n_0=700 ϵ = 5000	15	0	0
Surf method	t=0.008 n_0=1000 ϵ = 5000	9	0	6
	t=0.003 n_0=800 ϵ = 5000	13	0	2
	t=0.003 n_0=600 ϵ = 5000	15	0	0
Envelope match filter	t=3 n_0=1500 ϵ = 5500 g=400	15	0	0
	t=1 n_0=1000 ϵ = 5500 g=400	15	0	0
	t=1 n_0=700 ϵ = 5000 g=400	15	0	0

Tablica 5.5: „Mój jest ten kawałek podłogi v.1”



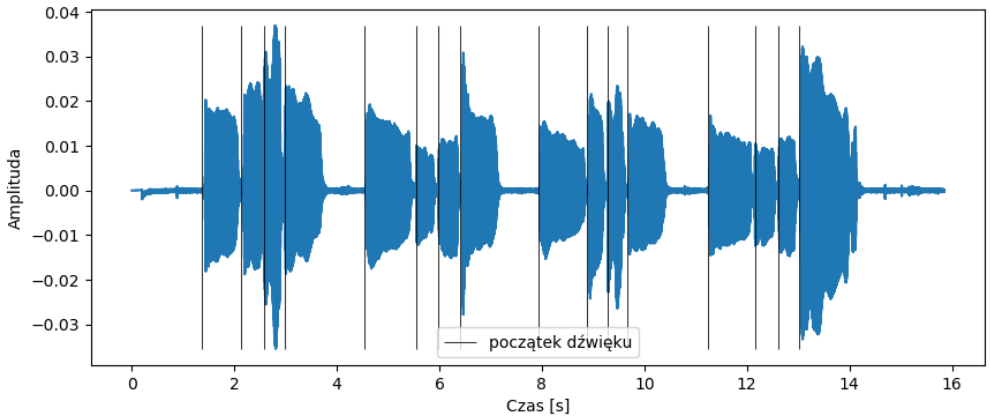
Rysunek 5.5: Envelope match filter, „Mój jest ten kawałek podłogi v.1”, $n_0 = 700$, $threshold = 1$, $epsilon = 5000$, gap = 400

W tej piosence, kolejny raz przy Surf method głównym problemem jest dopasowanie wielkości okna tak, aby onsety zaznaczać wystarczająco blisko faktycznego początku. Metoda Envelope match filter została w tym nagraniu testowana dla takich samych wartości n_0, ϵ i t jak Short-term energy method (g wszędzie wynosiło 400) i w przeciwieństwie do niej, Envelope match filter dla wszystkich z nich zwrócił prawidłowe odpowiedzi.

Mój jest ten kawałek podłogi v.2

Nazwa metody	Wartości stałych	Poprawne	Nadmierne	Niezaznaczone
Magnitude method	t=0.01 n_0=1000 ϵ=4500	8	0	6
	t=0.006 n_0=1000 ϵ=3500	16	2	0
	t=0.006 n_0=1000 ϵ=4500	16	0	0
Short-term energy method	t=0.005 n_0=400 ϵ=5500	13	2	3
	t=0.03 n_0=1300 ϵ=3800	16	3	0
	t=0.045 n_0=2030 ϵ=5500	16	0	0
Surf method	t=0.008 n_0=600 ϵ = 5000	0	0	16
	t=0.008 n_0=600 ϵ = 4500	1	0	15
	t=0.003 n_0=300 ϵ = 4500	12	0	4
	t=0.003 n_0=600 ϵ = 3000	12	1	4
	t=0.002 n_0=600 ϵ = 4000	15	0	1
	t=0.0018 n_0=600 ϵ = 4000	16	0	0
Envelope match filter	t=0.3 n_0=300 ϵ=6500 g=300	12	0	4
	t=0.3 n_0=1050 ϵ=4000 g=400	15	0	1
	t=0.5 n_0=500 ϵ=4000 g=300	16	0	0

Tablica 5.6: „Mój jest ten kawałek podłogi v.2”



Rysunek 5.6: Surf method, „Mój jest ten kawałek podłogi v.2”, $n_0 = 2500$, *threshold* = 6, *epsilon* = 6500

Podsumowując powyższe testy widać, że poszczególne metody mogą dawać prawidłowe wyniki dla całkiem różnych wartości stałych. Ich optymalne wartości zależą od tego jak szybko oraz jak głośno została zanucona/zaśpiewana piosenka. Widać jednak, że dla każdego z testów udało się dobrać stałe tak, że przynajmniej jeden z uzyskanych wyników był dobry.

5.2. Testy metod znajdujących wysokość dźwięku

Korzystając z programu porównującego wyniki wszystkich algorytmów wygenerowaliśmy dane do poniższych tabel. Nazwy metod zapisane zostały skrótowo i oznaczają kolejno: Funkcję autokorelacji (AF), Funkcję różnicy średnich wielkości (AMDF), Metodę cepstralną I (C1) i II (C2), Harmonic product spectrum (HP) i Trzecią największą wartość FFT(3BF). W przedostatnim wierszu każdej tabeli umieszczono listę wartości preferowanych (*Pref*) obliczonych za pomocą algorytmu wybierającego nuty, opisanego w rozdziale „Częstotliwości w dalszych krokach", to znaczy uwzględniamy wyniki czterech wyliczeń: C1, C2, HP i 3BF. Z kolei w ostatnim wierszu (oznaczonym *Ok*) znajdują się nuty, które wystąpiły najczęściej w danej kolumnie (pod uwagę zostały wzięte wartości ze wszystkich wierszy w tej kolumnie).

Gdy śliczna Panna v.1

Metoda	Dźwięki																					Ok
AF	A3	B3	C#4	D4	D4	E4	G4	F#4	E4	E4	D4	A#3	C4	D4	D#4	D#4	E4	G4	F#4	E4	E4	D4	22
AMDF	A3	B3	C#4	D4	D4	E4	G4	F#4	E4	E4	D4	A#3	C4	D4	D#4	D#4	D#4	G4	F#4	E4	E4	D4	21
C1	A3	B3	C#4	D4	D4	E4	G4	F#4	E4	E4	C#4	A3	C4	D4	C#4	D4	E4	F#4	F#4	E4	E4	C#4	17
C2	A3	B3	C#4	D#4	D4	E4	G4	F#4	E4	E4	D4	A#3	C4	D4	D#4	D#4	E4	F#4	F#4	E4	E4	D4	21
HP	A3	B3	C#4	D4	D4	E4	G4	F#4	E4	E4	D4	A3	C4	D4	D#4	D#4	E4	G4	G4	E4	E4	D4	20
3BF	A3	B3	C#4	D4	D4	D#4	G4	F#4	E4	E4	D4	A#3	C4	D4	D#4	D#4	E4	F#4	F4	E4	E4	D4	20
Pref	A3	B3	C#4	D4	D4	E4	G4	F#4	E4	E4	D4	A3	C4	D4	D#4	D#4	E4	F#4	F#4	E4	E4	D4	
Ok	A3	B3	C#4	D4	D4	E4	G4	F#4	E4	E4	D4	A#3	C4	D4	D#4	D#4	E4	F#4/ G4	F#4	E4	E4	D4	

Tablica 5.7: „Gdy śliczna panna v.1" – dźwięki

Gdy śliczna panna v.2

Metoda	Dźwięki																					Ok
AF	A#3	C#4	D#4	E4	E4	F#4	A4	G#4	F#4	F#4	E4	B3	C#4	D#4	E4	E4	F#4	A4	G#4	F#4	F#4	F4	22
AMDF	A#3	C#4	D#4	E4	E4	F#4	G#4	G#4	F#4	F#4	E4	B3	C#4	D#4	E4	E4	F#4	A4	G#4	F#4	F#4	F4	22
C1	A#3	C4	D#4	E4	E4	F#4	G#4	G#4	F#4	F#4	E4	B3	C#4	D#4	E4	E4	F#4	A4	F#4	F#4	F#4	F4	21
C2	A#3	C4	D#4	E4	E4	F#4	G#4	G#4	F#4	F#4	F4	B3	C#4	E4	E4	F4	F#4	A4	G4	F#4	F#4	F4	20
HP	A#2	C#4	D#2	E4	E4	F#4	A4	G#4	G4	F#4	F4	B3	C#4	E4	E4	E4	F#4	A#4	G#4	G4	F#4	F4	17
3BF	A#3	C4	D#4	E4	E4	F4	A4	A4	G4	F#4	F4	B3	D4	D#4	E4	E4	F#4	A4	G4	G4	G4	E4	15
Pref	A#3	C4	D#4	E4	E4	F#4	A4	G#4	G4	F#4	F4	B3	C#4	E4	E4	E4	F#4	A4	G4	G4	F#4	F4	
Ok	A#3	C4/C#4	D#4	E4	E4	F#4	A4/G#4	G#4	F#4	F#4	F4/E4	B3	C#4	D#4	E4	E4	F#4	A4	G#4	F#4	F#4	F4	

Tablica 5.8: „Gdy śliczna Panna v.2" – dźwięki

Szła dzieweczka v.1

Metoda	Dźwięki																									Ok
AF	B3	E4	G#4	F4	E4	D#4	E4	C4	B3	B3	A3	A#3	C4	A4	F4	A#3	A#3	G3	A3	A#3	F#4	D#4	B3	C4	A3	A#3	B3	24
AMDF	B3	E4	G#4	F4	E4	D#4	F4	C4	B3	B3	A3	A#3	C4	A4	E4	A#3	A#3	G#3	A3	A#3	F#4	D#4	B3	B3	A3	A#3	B3	27
C1	B3	E4	G#4	F4	E4	D#4	E4	C4	B3	B3	A3	A#3	C4	A4	E4	A3	A#3	C#5	A3	A#3	F4	D#4	B3	B3	A3	A#3	B3	23
C2	B3	E4	G#4	F4	E4	D#4	F4	C4	B3	C4	A3	A#3	C4	A4	F4	A#3	A#3	G#3	A3	A#3	F#4	D#4	B3	C4	A3	A#3	B3	24
HP	B3	E4	G#4	F#4	E4	D#4	F4	C4	B3	B3	A3	A#3	C4	A4	E4	A#3	A#3	F#3	A3	A#3	F#4	D#4	B3	B3	A3	A#3	B3	25
3BF	B3	E4	G#4	F4	E4	D#4	D4	C#4	B4	C4	A3	A#3	C4	A#4	E4	A#3	A#4	G3	A3	A#3	G4	D#4	B3	B3	A4	B3	B3	19
Pref	B3	E4	G#4	F4	E4	D#4	F4	C4	B3	B3	A3	A#3	C4	A4	E4	A#3	A#3	F#3	A3	A#3	F#4	D#4	B3	B3	A3	A#3	B3	
Ok	B3	E4	G#4	F4	E4	D#4	F4	C4	B3	B3	A3	A#3	C4	A4	E4	A#3	A#3	G3/G#3	A3	A#3	F#4	D#4	B3	B3	A3	A#3	B3	

Tablica 5.9: „Szła dzieweczka v.1" – dźwięki

Szła dzieweczka v.2

Metoda	Dźwięki																												Ok
AF	A#3	E4	A4	G4	F4	E4	F4	D4	C4	C4	A3	A#3	C4	G#4	E4	C4	A#3	A#3	G3	A3	A#3	F#4	D#4	C#4	B3	C4	G#4	F#4	E4	28
AMDF	A#3	E4	A4	G4	F4	E4	F4	D4	C4	C4	A3	A#3	C4	G#4	E4	C4	A#3	A#3	G#3	A3	A#3	F#4	D#4	C#4	B3	C4	G#4	F#4	E4	28
C1	A#3	F4	G#4	G4	F4	E4	F4	D#4	C4	C4	G#3	A#3	C4	G#4	E4	C4	A#3	A#3	D5	A3	A#3	F4	D#4	C#4	B3	C4	G4	F#4	E4	23
C2	A#3	F4	A3	G4	F4	E4	F4	D#4	C4	C4	A3	A#3	C4	G#4	E4	C4	A#3	A#3	G#3	A3	A#3	F#4	D#4	C#4	B3	C4	G4	F#4	E4	26
HP	A#3	F4	A4	G4	F4	E4	F4	D4	C3	C4	A3	A#3	C4	G#4	E4	C4	A#3	A#3	G3	A3	A#3	F#4	D#4	C#4	B3	C4	A4	F#4	E4	27
3BF	B3	F4	A4	G4	F4	F4	F4	D4	C4	C4	A3	A#3	C4	G#4	E4	C4	A#3	B3	G4	A3	A#3	F#4	D#4	C4	C4	C4	G#4	F#4	E4	23
Pref	A#3	F4	A4	G4	F4	E4	F4	D4	C4	C4	A3	A#3	C4	G#4	E4	C4	A#3	A#3	G3	A3	A#3	F#4	D#4	C#4	B3	C4	G4	F#4	E4	
Ok	A#3	F4	A4	G4	F4	E4	F4	D4	C4	C4	A3	A#3	C4	G#4	E4	C4	A#3	A#3	G3/G#3	A3	A#3	F#4	D#4	C#4	B3	C4	G#4	F#4	E4	

Tablica 5.10: „Szła dzieweczka v.2" – dźwięki

Mój jest ten kawałek podłogi v.1

Metoda	Dźwięki															Ok
AF	G#3	A3	A#3	G3	G#3	D#4	D4	G#3	G#3	A3	B3	G#3	G#3	D#4	D4	14
AMDF	G#3	A3	B3	G#3	G#3	D#4	D4	G#3	G#3	A3	B3	G#3	G#3	D#4	D4	13
C1	D5	A3	B3	G3	G3	D#4	D4	G3	G3	A3	A#3	G3	G3	D#4	D4	11
C2	A3	A3	B3	G3	G3	D#4	D4	G#3	G3	A3	B3	G3	G#3	D#4	D#4	13
HP	G3	A3	A#3	G3	G3	D#4	D4	G#3	G3	A3	B3	G#3	G#3	E4	D4	13
3BF	G#3	A4	A#3	G3	G3	A#5	D4	G#3	G#3	A#3	B3	G3	G#3	E4	D#4	11
Pref	G3	A3	A#3	G3	G3	D#4	D4	G#3	G3	A3	B3	G3	G#3	E4	D4	
Ok	G#3	A3	A#3/B3	G3	G3	D#4	D4	G#3	G#3/G3	A3	B3	G#3/G3	G#3	D#4	D4	

Tablica 5.11: „Mój jest ten kawałek podłogi v.1" – dźwięki

Mój jest ten kawałek podłogi v.2

Metoda	Dźwięki																Ok
AF	G3	G#3	A3	G3	G3	D#4	D4	G3	G3	G#3	A3	G3	G3	D4	D4	C4	16
AMDF	G#3	G#3	A3	G#3	G#3	D#4	D#4	G#3	G#3	G#3	A3	G#3	G#3	D4	D4	C4	8
C1	A#4	G#3	C5	G#3	A#4	D#4	D4	C5	A#4	G#3	A#4	C#5	C5	D#4	D4	C4	6
C2	A3	G#3	A3	A3	G#3	D#4	D4	G3	A3	G#3	G#3	A3	C4	D4	D4	C4	9
HP	G3	G#2	A3	G3	G3	D#4	D3	G3	G3	G#3	G#3	G3	G3	D4	D4	B3	12
3BF	G3	G#4	A#3	G3	G3	D4	A#3	G3	G3	G#4	A3	G3	G3	D4	D4	C4	12
Pref	G3	G#3	A3	G3	G3	D#4	D4	G3	G3	G#3	G#3	G3	G3	D4	D4	C4	
Ok	G3	G#3	A3	G3	G3	D#4	D4	G3	G3	G#3	A3	G3	G3	D4	D4	C4	

Tablica 5.12: „Mój jest ten kawałek podłogi" v.2 – dźwięki

Nie sprawdzamy zgodności z oryginalnymi nutami lub odpowiednim ich przesunięciem, ponieważ w testach nie oceniamy zgodności śpiewu człowieka ze wzorowym wykonaniem piosenkarza, a raczej zdolność algorytmu do identyfikowania wysokości

śpiewanego dźwięku. Ponieważ te algorytmy są niezależne i różne od siebie możemy uznać, że skoro większość zwraca jakiś wynik, to z dużym prawdopodobieństwem jest to wynik prawdziwy. Za prawidłową nutę uznajemy więc tę, która została zwrócona przez największą liczbę algorytmów (są one zawarte w wierszu *Ok*). Bazując na tej definicji poprawności nuty możemy teraz ocenić pojedynczy algorytm – jeżeli w odniesieniu do prawidłowych nut algorytm zwraca w znacznym stopniu izomorficzne wyniki, to możemy uznać go za działający poprawnie. W ostatniej kolumnie każdej z tabel mamy dla każdego algorytmu informację ile nut, które zwrócił jest takich samych jak te z wiersza *Ok* (czyli tych prawidłowych).

Widzimy, że każdy algorytm zdobył dobre wyniki w kolumnie punktów. Na prowadzenie wysuwa się Funkcja autokorelacji – wyniki dla niej w prawie każdej piosence są najwyższe. Można więc stwierdzić, że do wyboru preferowanych nut do kolejnego kroku dobrze byłoby uwzględniać jej wyniki. Jednak po początkowych testach algorytmów Autokorelacja nie została wzięta pod uwagę do „preferowanych nut" z powodu kwadratowej złożoności osiąganej przez nią bez wprowadzonego przez nas przyspieszenia.

5.3. Testy metod dopasowujących melodie

Mając obliczone częstotliwości podstawowe kolejnych dźwięków możemy przejść do obliczania ich dopasowania do piosenek w bazie. W tym celu wzięliśmy z internetu po dwa różne nagrania do każdego z nuconych utworów. Były to, kolejno według poniższych tabel, „Gdy śliczna panna" śpiewane męskim głosem i z głośnym podkładem muzycznym, „Gdy śliczna panna" grane na pianinie, „Mój kawałek podłogi" z oryginalnymi trąbkami z początku utworu, „Mój kawałek podłogi" grany na pianinie, „Szła dzieweczka" śpiewane przez dwa dziewczęce głosy i „Szła dzieweczka" grane na pianinie na dwie ręce.

Przed testami dla skalowania liniowego, zostały sprawdzone wyniki podejścia wykorzystującego programowanie dynamiczne. 5.13 to tabela wyników dla porównań nucenia „Szła dzieweczka do laseczka v.2" mającego 29 nut z pozostałymi nuceniami z bazy. Przy każdym wyniku znajduje się informacja ile nut zostało uzyskanych z formatu MIDI dla każdego z porównywanych nagrań. 5.14 jest tabelą zawierającą wyniki dopasowań nucenia „Szła dzieweczka do laseczka v.2", jednak tutaj z nagraniami, które zostały pobrane z serwisu Youtube.

Utwór	Liczba nut	Wynik
GdyŚlicznaPanna v.2	70	7
GdyŚlicznaPanna v.1	152	11
KawałekPodłogi v.2	24	8
KawałekPodłogi v.1	117	2
SzłaDzieweczka v.2	55	36
SzłaDzieweczka v.1	113	15

Tablica 5.13: „Szła dzieweczka do laseczka v.2", 29 nut – programowanie dynamiczne, dopasowanie z pozostałymi nuceniami

Utwór	Liczba nut	Wynik
GdyŚlicznaPanna1	1468	1
GdyŚlicznaPanna2	41	3
KawałekPodłogi1	512	0
KawałekPodłogi2	144	0
SzłaDzieweczka1	327	0
SzłaDzieweczka2	84	0

Tablica 5.14: „Szła dzieweczka do laseczka v.2", 29 nut – programowanie dynamiczne, dopasowanie z nagraniami pobranymi z Youtube

Widać, że algorytm nie radzi sobie z dopasowaniami do utworów bardziej złożonych niż samo nucenie. O wiele lepsze wyniki dopasowań do nagrań z Youtube zostały uzyskane dla testów skalowania liniowego, które znajdują się poniżej.

Gdy śliczna Panna v.1

Utwór	Wynik	Współczynnik	Średnia
GdyŚlicznaPanna1	14	0.5	9,5
GdyŚlicznaPanna2	5	0.75	
KawałekPodłogi1	4	0.5	4
KawałekPodłogi2	4	0.75	
SzłaDzieweczka1	4	0.5	8
SzłaDzieweczka2	12	0.5	

Tablica 5.15: „Gdy śliczna Panna v.1" – skalowanie liniowe

Gdy śliczna Panna v.2

Utwór	Wynik	Współczynnik	Średnia
GdyŚlicznaPanna1	18	1.0	11
GdyŚlicznaPanna2	4	1.0	
KawałekPodłogi1	4	1.0	4
KawałekPodłogi2	4	1.25	
SzłaDzieweczka1	5	0.5	8
SzłaDzieweczka2	11	0.5	

Tablica 5.16: „Gdy śliczna Panna v.2" – skalowanie liniowe

Szła dzieweczka v.1

Utwór	Wynik	Współczynnik	Średnia
GdyŚlicznaPanna1	16	0.5	11,5
GdyŚlicznaPanna2	5	0.5	
KawałekPodłogi1	6	0.5	5
KawałekPodłogi2	4	1.5	
SzłaDzieweczka1	15	0.5	12,5
SzłaDzieweczka2	10	0.5	

Tablica 5.17: „Szła dzieweczka v.1" – skalowanie liniowe

Szła dzieweczka v.2

Utwór	Wynik	Współczynnik	Średnia
GdyŚlicznaPanna1	15	0.5	9
GdyŚlicznaPanna2	3	0.75	
KawałekPodłogi1	4	0.5	5
KawałekPodłogi2	6	0.75	
SzłaDzieweczka1	7	0.5	9,5
SzłaDzieweczka2	12	0.5	

Tablica 5.18: „Szła dzieweczka v.2" – skalowanie liniowe

Mój jest ten kawałek podłogi v.1

Utwór	Wynik	Współczynnik	Średnia
GdyŚliczna Panna1	5	0.5	3
GdyŚliczna Panna2	1	0.5	
KawałekPodłogi1	6	0.5	6,5
KawałekPodłogi2	7	0.75	
SzłaDzieweczka1	5	0.5	5,5
SzłaDzieweczka2	6	1.25	

Tablica 5.19: „Mój jest ten kawałek podłogi v.1" – skalowanie liniowe

Mój jest ten kawałek podłogi v.2

Utwór	Wynik	Współczynnik	Średnia
GdyŚliczna Panna1	6	1.0	3,5
GdyŚliczna Panna2	1	0.5	
KawałekPodłogi1	5	0.5	7
KawałekPodłogi2	9	0.75	
SzłaDzieweczka1	3	1.25	5
SzłaDzieweczka2	7	1.25	

Tablica 5.20: „Mój jest ten kawałek podłogi v.2" – skalowanie liniowe

Algorytm nie zawsze działa idealnie w przypadku jednostkowego porównywania piosenek. Najlepiej działa przy nuconych dłuższych fragmentach oraz przy oryginalnych utworach z niedużą liczbą dźwięków w tle. Jednak, nawet przy piosenkach z głośnym tłem, dla wszystkich sprawdzanych nuceń średnia wartość wyników najwyższa jest dla faktycznych oryginałów.

Rozdział 6.

Wnioski

Powyższe testy pokazują, że algorytmy znajdujące początki dźwięków są w stanie dawać bardzo dobre wyniki dla odpowiednio ustawionych wartości *okna*, *progu*, *epsilon* oraz – w przypadku Envelope match filter – dla wartości *gap*. Optymalne wartości tych parametrów zależą od tego jak szybko jest nucona dana piosenka, jak wysoko. Algorytmy znajdujące wysokość dźwięku, mimo że są od siebie niezależne, dają dla danego nagrania bardzo podobne wyniki. W przypadku algorytmów dopasowujących melodię, algorytm skalowania liniowego dał dobre wyniki. Dla każdego z przeprowadzonych testów średnia wyników zwróconych dla różnych wersji tego samego utworu znajdujących się w bazie była faktycznie największa dla szukanej przy tym teście piosenki. Programowanie dynamiczne zwróciło dobre wyniki dla dopasowywania naszego nucenia z innymi nuceniami z bazy, jednak porównując je z innymi nagraniami niebędącymi nuceniami algorytm daje słabe wyniki.

Po całej powyższej analizie widać, że zagadnienie porównywania utworów jest trudne i skomplikowane, jednak możliwe jest uzyskanie satysfakcjonujących rezultatów. Wykorzystując różne podejścia na kolejnych etapach analizy dźwięku coraz bardziej przybliżamy się do odpowiedzi na pytanie o podobieństwo dwóch nagrań. Prawdopodobnie, przy większej liczbie usprawnień obecnych algorytmów, wprowadzeniu kolejnych metod do dopasowywania melodii oraz rozszerzeniu bazy moglibyśmy dostać wyniki jeszcze bliższe ideału i w rezultacie uzyskać odpowiedź na pytanie „Jaka piosenka została zanucona?”.

Bibliografia

[1] Autokorelacja — Wikipedia, the free encyclopedia. https://pl.wikipedia.org/wiki/Autokorelacja.

[2] Forum stackoverflow, mark lines on the plot. https://stackoverflow.com/questions/41824662/how-to-plot-a-dot-each-time-at-the-point-the-mouse-is-clicked-in-matplotlib.

[3] Improved onset detection algorithm based on fractional power envelope match filter. https://www.eurasip.org/Proceedings/Eusipco/Eusipco2011/papers/1569422835.pdf.

[4] Mido library midi files. https://mido.readthedocs.io/en/latest/midi_files.html.

[5] Note to frequency converter. https://gist.github.com/CGrassin/26a1fdf4fc5de788da9b376ff717516e#file-note_to_freq-py.

[6] Pitch detection methods. https://sound.eti.pg.gda.pl/student/eim/synteza/leszczyna/index_ang.htm.

[7] Splot(analiza matematyczna) — Wikipedia, the free encyclopedia. https://pl.wikipedia.org/wiki/Splot_(analiza_matematyczna).

[8] Virtual piano. https://virtualpiano.net/.

[9] Forum stackoverflow, converting midi ticks to actual playback seconds. https://stackoverflow.com/questions/2038313/converting-midi-ticks-to-actual-playback-seconds, 10 stycznia 2010.

[10] Wlazł kotek na płotek – nuty literowe. https://nutyliterowe.pl/wszystkie/wlazl-kotek-na-plotek, 11 lutego 2019.

[11] Forum redit, post z pytaniem. https://www.reddit.com/r/tipofmytongue/comments/3pzrmj/tomtsong_song_that_goes_la_la_la_la_la_la_la_la/, 24 października 2015.

[12] Forum muzyka z reklam, post z pytaniem. https://muzykazreklam.pl/53499-lalalala-la-lalala-la-aaaaa-i-pisk, 3 marca 2017.

[13] Chiao-Wei Lin. Query by singing and humming, 2015.

[14] Thanh Vo and Hideyuki Sawada. Intoning speech performance of the talking robot for vietnamese language case. grudzień 2018.