POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Elektronika i Telekomunikacja
SPECJALNOŚĆ: Inżynieria akustyczna

PRACA DYPLOMOWA INŻYNIERSKA

Realizacja procedury detekcji stresu w głosie

Implementation of Stress Detection Procedure

AUTOR:
Piotr Lechowicz

PROWADZĄCY PRACĘ:
dr inż. Piotr Staroniewicz

OCENA PRACY:

WROCŁAW 2014

Spis treści

1. Wstęp

Stres towarzyszył człowiekowi od najwcześniejszych lat powstawania gatunku ludzkiego. Początkowo pełnił funkcję obronną — mobilizował organizm do walki z wrogiem lub do ucieczki. Obecnie, mimo że rzadko zdarzają się sytuacje bezpośredniego zagrożenia życia, stres jest nieodłącznym towarzyszem człowieka. Stresują nas sytuacje życiowe takie jak: utrata bliskiej osoby, zmiana sytuacji życiowej, zmiana pracy, małżeństwo, przyjście na świat dziecka, konflikt z osobą dla nas znaczącą, egzamin itp. Oprócz tak oczywistych czynników wywołujących stres istnieją również sytuacje bardziej finezyjne, gdy nie jest wprawdzie zagrożony byt człowieka, ale jest zagrożony jego wewnętrzny wizerunek — poczucie własnej wartości i obraz siebie jako uczciwego człowieka. Do sytuacji takiej dochodzi, gdy człowiek kłamie lub próbuje zataić prawdę. W większości przypadków kłamstwo dotyczy spraw nieistotnych z punktu widzenia dobra społecznego, jednakże czasem od słów człowieka może zależeć życie, zdrowie lub dobra materialne innej osoby. Wówczas możliwość weryfikacji prawdomówności staje się rzeczą konieczną. Dlatego też wraz z rozwojem technologii są opracowywane coraz to nowsze możliwości detekcji kłamstwa. Jedną spośród dostępnych technik jest analiza zawartości stresu w głosie, nazywana VSA[1].

1.1. Cel pracy

Celem mojej pracy było przygotowanie aplikacji komputerowej realizującej procedurę detekcji stresu w głosie. Do sprawdzenia skuteczności działania zaimplementowanego algorytmu zostały wykorzystane dwie bazy nagrań, przygotowane w ramach prac dyplomowych Zbigniewa Kurka i Anny Tarnawskiej. Następnie wyniki zostały porównane ze skutecznością algorytmów detekcji stresu wykonanych w ramach prac magisterskich Radosława Rutkowskiego oraz Piotra Fleszara.

1.2. Rys historyczny

W 1970 roku trzech emerytowanych amerykańskich oficerów — Alan Bell, Bill Ford, Charles McQuiston — stworzyło firmę CIS. Każdy z nich specjalizował się w innej dziedzinie, Bell w kontrwywiadzie, Ford w elektronice, a McQuiston w poligrafii. Wynaleźli oni elektroniczne urządzenie, na którym nagrywali głos ludzki, spowalniali go czterokrotnie, przepuszczali przez kilka filtrów dolnoprzepustowych i ostatecznie rejestrowali na papierze za pomocą wydruku EKG. Nazwali swoje urządzenie Psychological Stress Evaluator (PSE). Było to pierwsze urządzenie typu VSA [3].

W związku z odniesionym sukcesem przez Bella, Forda i McQuistona, inne firmy zainteresowały się tą gałęzią wiedzy. W 1988 roku firma National Institue of Truth Verification wyprodukowała analogowe urządzenie CVSA. Bazowało ono na podobnym algorytmie jak urządzenie firmy CIS. Dziewięć lat później po praz pierwszy zaimplementowano je jako program komputerowy. W roku 2007 powstała ulepszona wersja systemu dodająca poszerzony zakres funkcjonalności. Innym programem, który pojawił się na rynku w roku 2001, był X13-VSA firmy X13-VSA Ltd. Początkowo miał służyć jedynie do celów militarnych, jednakże wkrótce stał się jednym z najtańszych systemów na rynku. Istnieje wiele innych produktów do zastosowań nieprofesjonalnych. Jednym z przykładów jest niekomercyjny program LiarLiar przygotowany na platformę operacyjną Linux.

[1] VSA (Voice Stress Analysis) – z ang. analiza stresu w głosie.

Ponieważ kod źródłowy programu jest ogólnie dostępny, może on być udoskonalany przez inne osoby [10].

Poza technikami VSA istnieje wiele innych dostępnych metod detekcji kłamstwa, takich jak wariograf lub różnego rodzaju metody oparte o badanie mózgu. Jednakże niski koszt przeprowadzenia badania, brak potrzeby posiadania specjalistycznego sprzętu i brak konieczności specjalistycznego przeszkolenia osoby weryfikującej podejrzanego powoduje, że VSA jest stale rozwijane.

2. Wprowadzenie

2.1.Podstawy teoretyczne techniki VSA

Większość sprzedawców urządzeń posługujących się technologią VSA nie chce zdradzać szczegółów działania ich systemu. Jednakże za wszystkimi implementacjami stoją te same podłoża teoretyczne. W trakcie powstawania mowy poza normalnymi drganiami mięśni krtani występują też drgania o częstotliwościach infradźwiękowych, zwane *mikrodrżeniami*, bezpośrednio pojawiające się w głosie człowieka. W momentach stresu, szczególnie gdy człowiek jest wystawiony na niebezpieczeństwo, ciało przygotowuje się do walki lub ucieczki zwiększając aktywność mięśni. Przejawia się to zmianą częstotliwości mikrodrżeń [4].

Podstawy teoretyczne mikrodżeń, wykorzystywanych w technice VSA, opublikował w 1957 roku brytyjski filozof Olaf Lippold. Falowanie mięśni, znane też pod pojęciem fizjologicznych drgań, zostało odkryte dużo wcześniej, ale było utożsamiane z zupełnie innymi przyczynami – uderzeniami serca lub falami alfa mózgu. Dopiero badania Lippolda pozwoliły określić faktyczne źródło mikrodrżenia. Jest nim serwomechanizm w organizmie i sprzężenie zwrotne, kontrolujące napięcie i długość mięśnia [8]. Pierwotnie badania były przeprowadzone na mięśniach odpowiedzialnych za poruszanie się — głównie mięśnie nóg, rąk i palców. Jednakże późniejsze badania uogólniły teorię Lippolda także na zachowanie mięśni dotyczących produkcji mowy. Wszystkie mięśnie wliczając w to mięśnie krtani drgają naturalnie z częstotliwością pomiędzy 8, a 12 Hz. W przypadku wystąpienia czynników wywołujących stres organizm zmienia częstotliwość mikrodrżeń, co może być zauważone bezpośrednio w wyprodukowanym przez człowieka sygnale mowy [7].

2.2.Zalety metody VSA

Metody VSA posiadają przewagę nad innymi sposobami detekcji kłamstwa z powodu niektórych właściwości:

- Niski koszt przeprowadzenia badania. W celu realizacji tego procesu potrzebny jest jedynie komputer i mikrofon, bez żadnego drogiego, specjalistycznego sprzętu.
- Krótki czas trwania badania. Niektóre metody potrzebują wielogodzinnych badań z potrzebą zadawania pytań nieistotnych, kontrolnych i istotnych dla badanej sprawy.
- Nie potrzeba szkolić wykwalifikowanej osoby do przeprowadzenia badania. Metoda VSA w jasny sposób określa wykrycie stresu w głosie.
- Możliwość analizy nagrań zarchiwizowanych.
- Nieinwazyjność metody. Nie trzeba umieszczać na badanym żadnej aparatury pomiarowej, jak w przypadku badania wariografem. Podejrzany może nie mieć świadomości przeprowadzania badania.
- Nie trzeba wykonywać badania w specjalnie odizolowanym pomieszczeniu lub też w pomieszczeniu z trudną w transporcie aparaturą. Badanie może być wykonane zdalnie poprzez umieszczenie mikrofonu w środowisku badanego.
- Implementacja technik VSA jest dużo tańsza od innych metod. Tańszy jest również koszt prób dalszego rozwoju techniki.

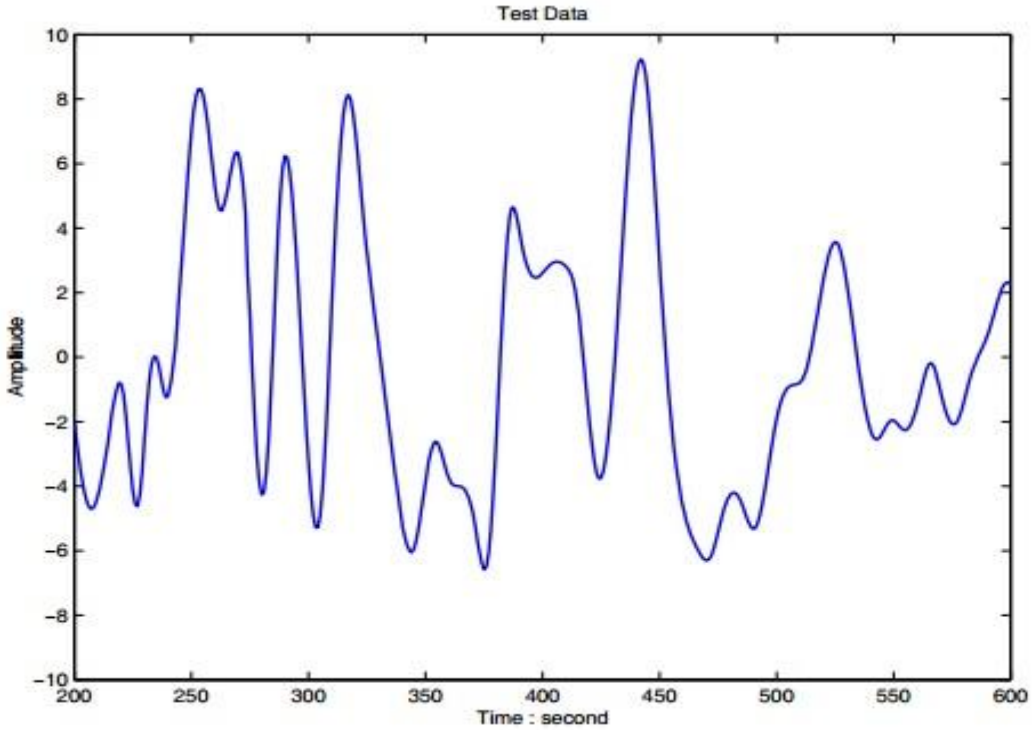
2.3.Algorytm EMD

Kluczową częścią technik VSA jest wydobycie mikrodrżenia z badanej próbki. Do otrzymania częstotliwości składowych sygnału może posłużyć wynaleziona przez N. E. Huanga metoda empirycznej dekompozycji modów funkcji nazywana EMD (Empirical Mode Decomposition). Zaproponowana przez Huanga metoda służy do analizy sygnałów niestacjonarnych i nieliniowych. Dekompozycja bazuje na prostym założeniu, że każdy sygnał jest złożeniem różnych prostych wewnętrznych modów oscylacji. W każdym momencie sygnał może mieć wiele różnych współistniejących modów. Po zsumowaniu dają w rezultacie oryginalny sygnał. Każda ze składowych oscylacji jest reprezentowana przez funkcję IMF (Intrinsic Mode Function), która jest definiowana w następujący sposób:

- liczba miejsc zerowych w sygnale i ekstremów musi się równać sobie albo różnić co najwyżej o jeden;
- w każdym punkcie średnia z obwiedni wykreślonych na podstawie górnych i dolnych ekstremów jest równa zero.

Funkcje IMF reprezentują składowe oscylacje sygnału, ale jest to pojęcie dużo bardziej ogólne od funkcji harmonicznych. Częstotliwość i amplituda funkcji IMF są zmiennymi wyrażonymi w funkcji czasu [5].

Technika EMD jest opisana za pomocą algorytmu, a nie analitycznej formuły, która pozwoliłaby na teoretyczną analizę algorytmu. Poniżej przedstawiono działanie algorytmu.

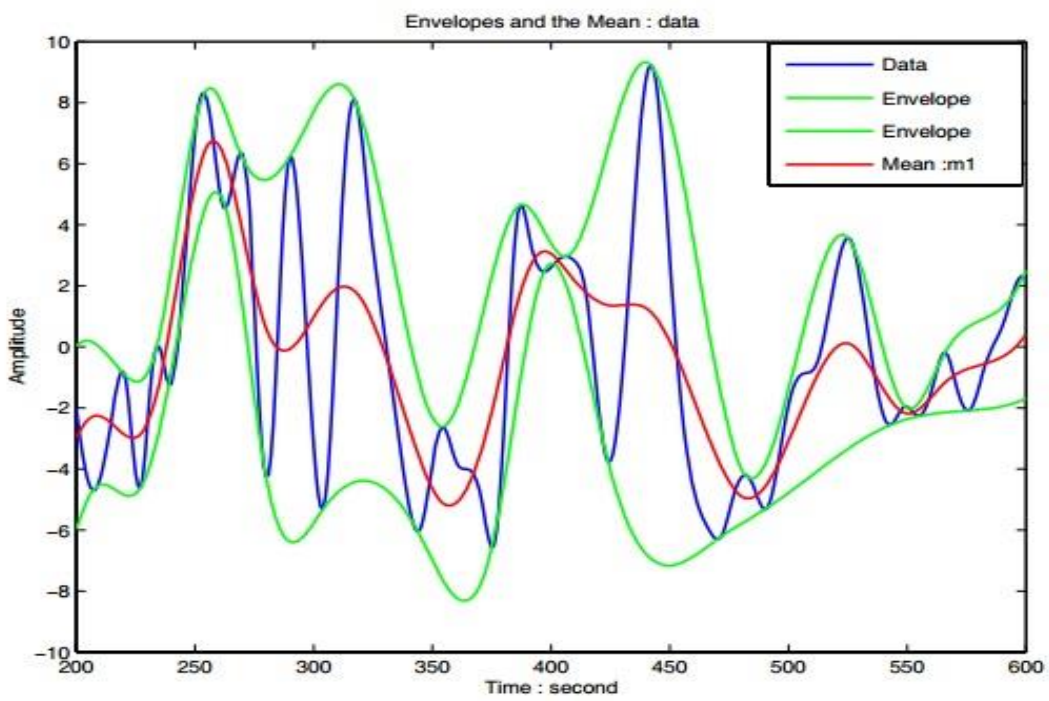


Rys. 1. Sygnał wejściowy do analizy EMD [5]

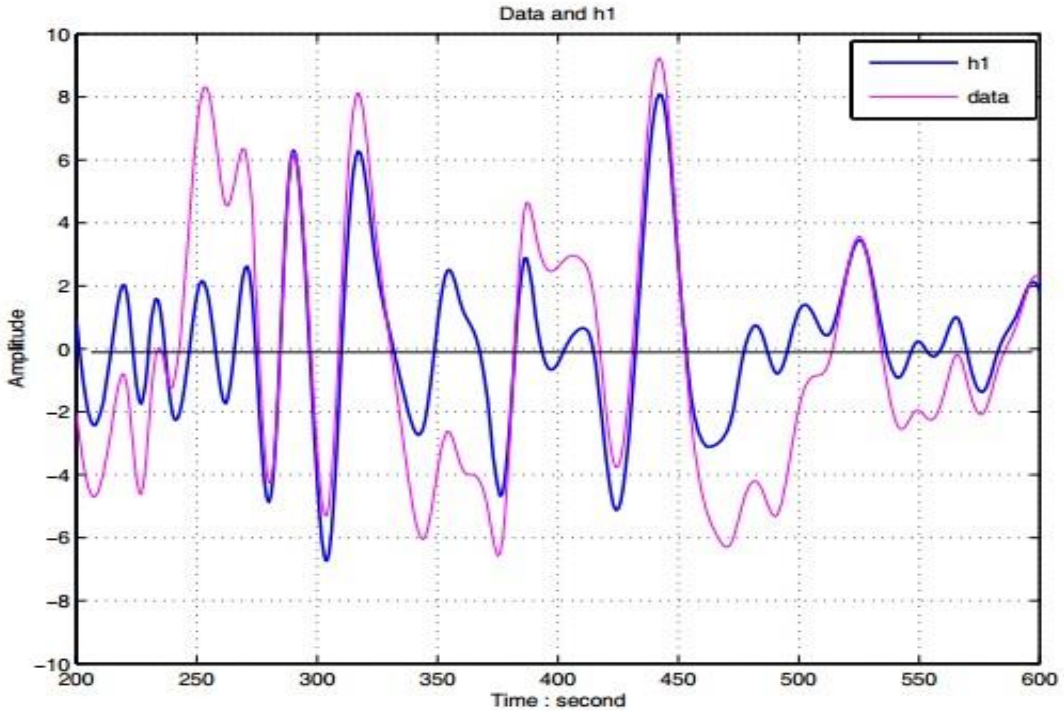
Pierwszym krokiem jest wyznaczenie wszystkich lokalnych ekstremów z sygnału wejściowego $x(t)$ (rys. 1). Następnie należy połączyć lokalne maksima za pomocą interpolacji funkcjami sklejanymi trzeciego rzędu. W analogiczny sposób należy postąpić z łączeniem lokalnych minimów. Cały sygnał powinien zawierać się pomiędzy górną i dolną

obiednią. Następnie należy wyliczyć średnią z obwiedni oznaczoną jako m_1 (rys. 2). Różnica pomiędzy sygnałem i średnią z obwiedni jest pierwszym komponentem h_1 (rys. 3.).

$$h_1 = x(t) - m_1$$

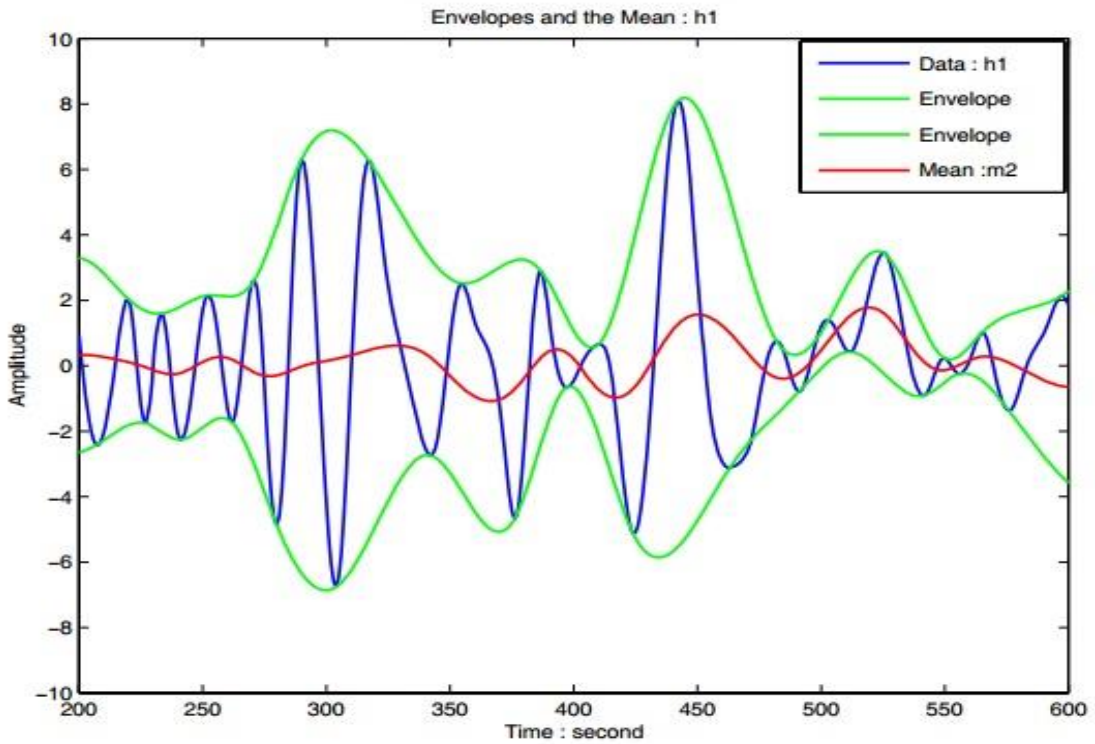


Rys. 2. Wykres przedstawiający sygnał wejściowy, jego górną i dolną obwiednię, oraz średnią z obwiedni (*sygnał wejściowy — kolor niebieski; górna i dolna obwiednia — kolor zielony; średnia z obu obwiedni — kolor czerwony*) [5]



Rys. 3. Wykres przedstawiający sygnał wejściowy i obliczony pierwszy komponent (*sygnał wejściowy — kolor fioletowy; pierwszy komponent — kolor niebieski*) [5]

W przypadku idealnym h_1 mogłoby spełniać wymagania funkcji IMF. Jednakże w rzeczywistości rzadko tak się zdarza. Otrzymanie pierwszego komponentu jest nazywane pierwszym przesiewem.



Rys. 4. Powtórzenie przesiewu dla pierwszego komponentu; na wykresie znajdują się górna i dolna obwiednia oraz sygnał będący średnią z obwiedni (*pierwszy komponent — kolor niebieski, górna i dolna obwiednia — kolor zielony, średnia z obwiedni —kolor czerwony*) [5]

Następny komponent otrzymuje się poprzez odjęcie od otrzymanego komponentu średniej z jego obwiedni (rys. 4). W kolejnych iteracjach jest powtarzany proces przesiewania (rys. 5) aż do otrzymania komponentu spełniającego kryteria. Jeżeli otrzymany sygnał spełnia przyjęte założenia, jest zapisywany jako pierwsza funkcja IMF.

Matematyczne przedstawienie drugiej iteracji wygląda następująco

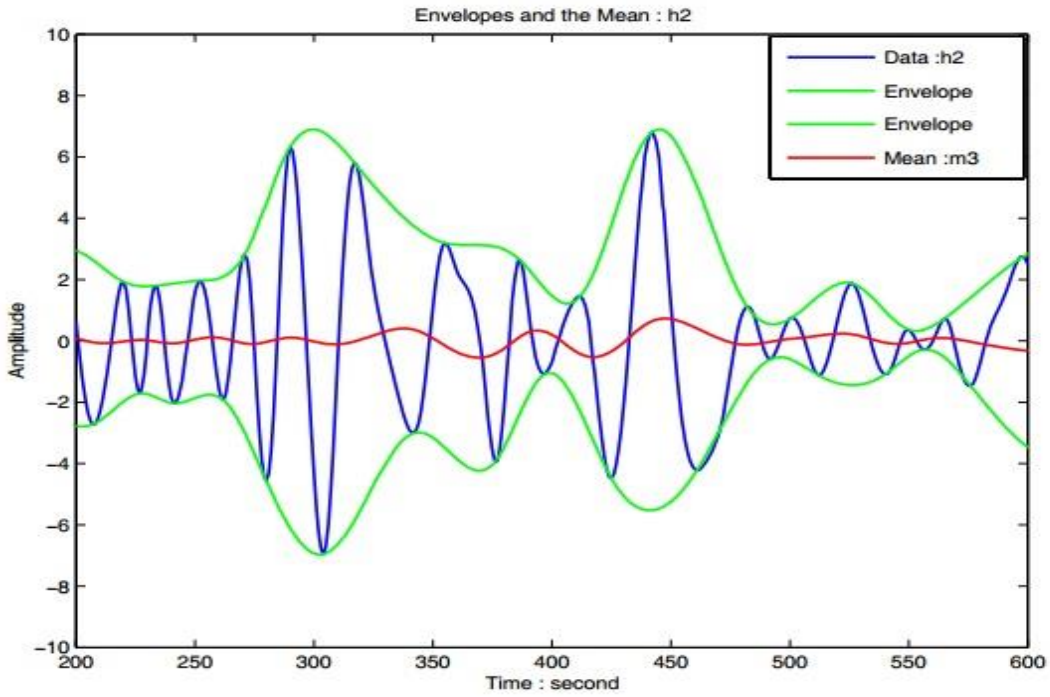
$$h_{11} = h_1 - m_{11}.$$

Kolejne przesiewy można opisać wzorem

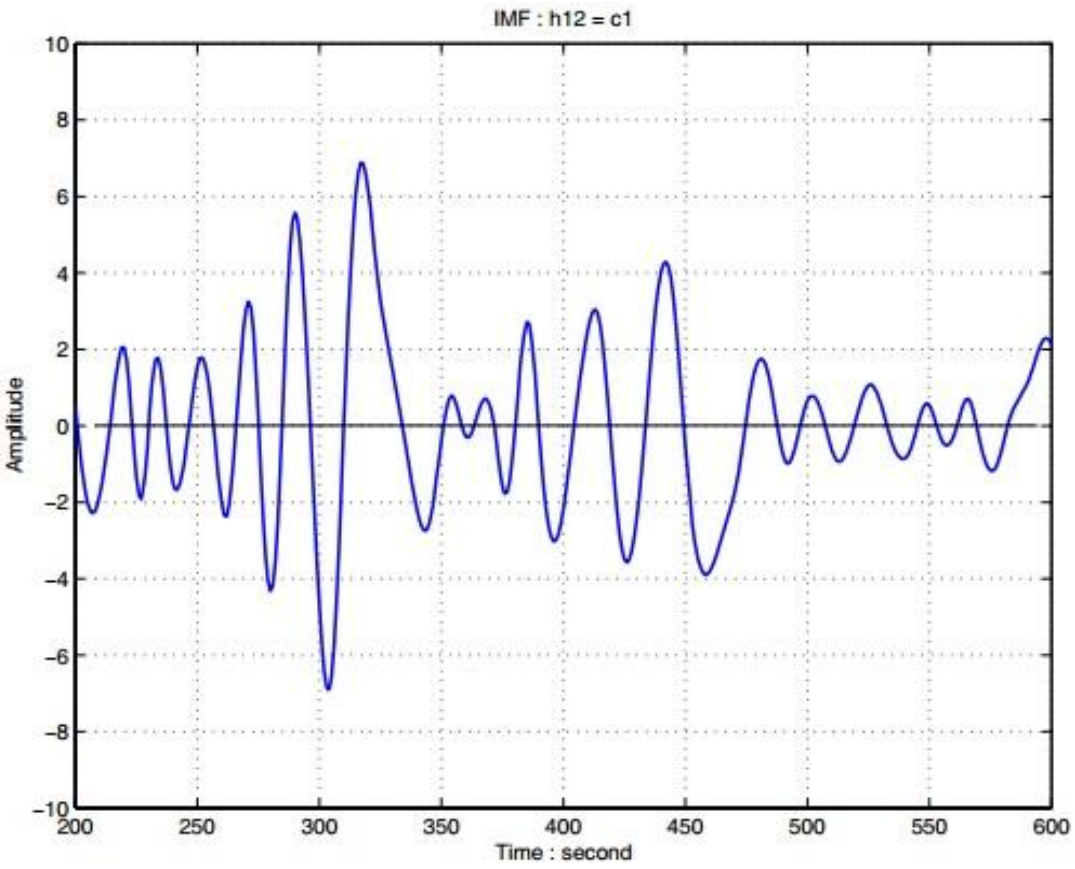
$$h_{1k} = h_{1(k-1)} - m_{1k}.$$

W pewnym momencie zostaje otrzymana funkcja IMF (rys. 6) wyrażona jako

$$c_1 = h_{1k}.$$



Rys. 5. Powtórzenie przesiewu dla drugiego komponentu; na wykresie znajdują się także jego górna i dolna obwiednie oraz sygnał będący średnią z obwiedni (*pierwszy komponent — kolor niebieski, górna i dolna obwiednia — kolor zielony, średnia z obwiedni —kolor czerwony*) [5]



Rys. 6. Pierwsza funkcja IMF otrzymana po dwunastu krokach przesiewu [5]

Istotnym staje się określenie kryterium stopu wykonywania przesiewów, ponieważ średnia z obwiedni tylko teoretycznie będzie idealnie płaska. W niniejszej pracy zostało wykorzystane kryterium zaproponowane przez Huanga — odchylenie standardowe między dwoma kolejnymi przesiewami, określone wzorem:

$$SD_k = \frac{\sum_{t=0}^{T} |h_{k-1}(t) - h_k(t)|^2}{\sum_{t=0}^{T} h_{k-1}^2},$$

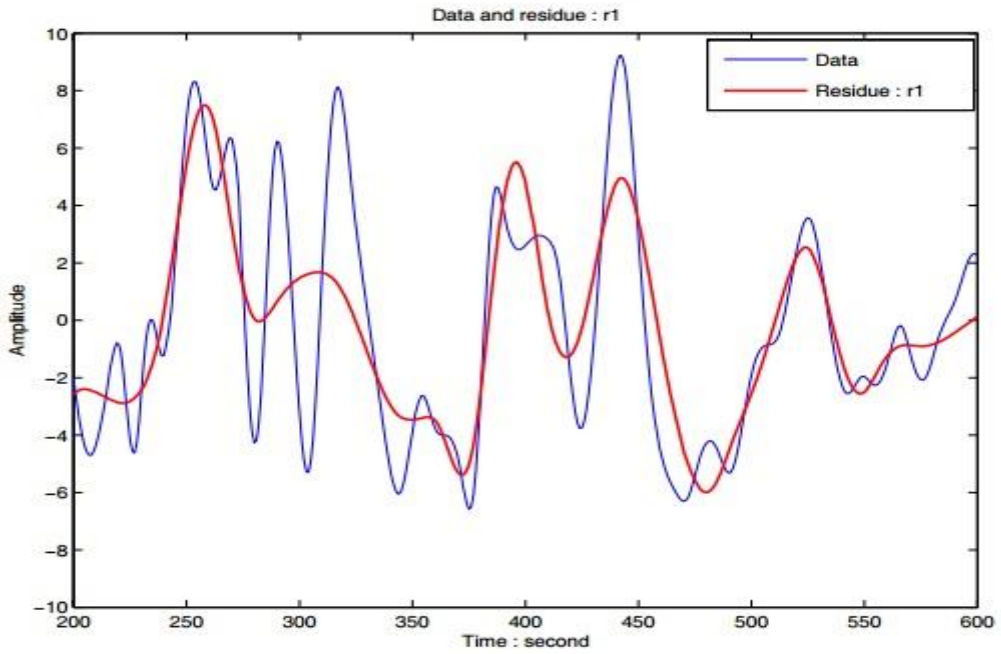
gdzie SD_k jest odchyleniem standardowym pomiędzy k-tym, a k–1-szym przesiewem; h_k jest k-tym komponentem; h_{k-1} jest k–1-szym komponentem.

Wartość odchylenia standardowego w literaturze została określona jako „mała", jednakże nie została sprecyzowana. Żadna wartość nie gwarantuje, że funkcja będzie miała taką samą liczbę miejsc zerowych i ekstremów (albo różniącą się o jeden). Wartość odchylenia standardowego, decydującego o kryterium stopu, w zrealizowanej aplikacji ma wartość dobraną empirycznie, stanowiącą kompromis pomiędzy dokładnością, a wymaganym czasem obliczeń.

Zakładając, że kryterium stopu zostało osiągnięte, otrzymaną funkcję IMF należy odseparować od sygnału wejściowego przeznaczonego do analizy:

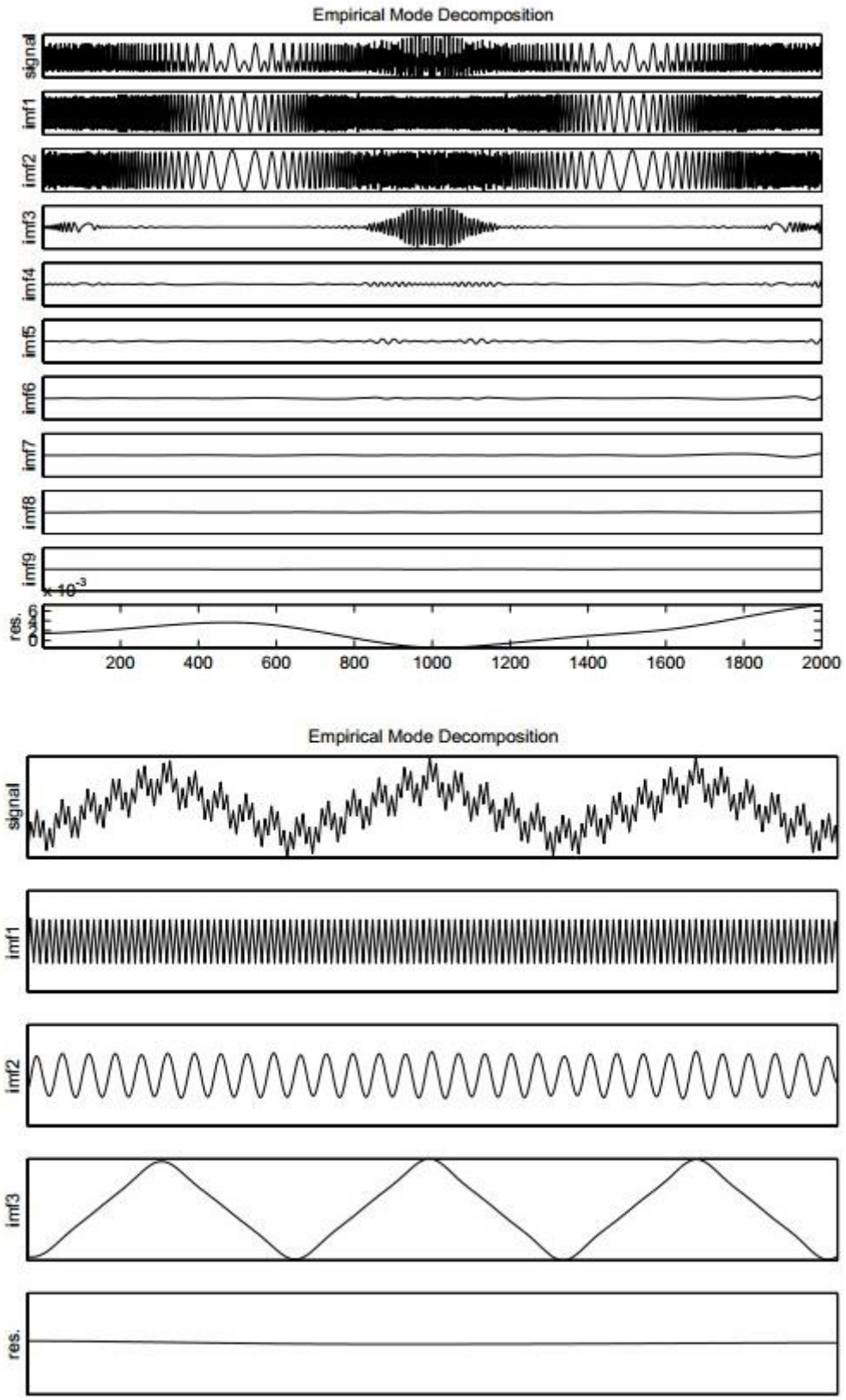
$$r_1 = x(t) - c_1.$$

Tak otrzymana funkcja jest nazywana pierwszym residuum (rys. 7).



Rys. 7. Wykres przedstawiający otrzymany sygnał po odjęciu sygnału wejściowego i pierwszej funkcji IMF (*sygnał wejściowy — kolor niebieski, residuum — kolor czerwony)* [5]

Cały proces odsiewania funkcji IMF jest powtarzany na pierwszym residuum i na następnych do momentu kiedy nie jest możliwe otrzymanie kolejnej funkcji IMF — tj. funkcja jest monotoniczna albo posiada tylko jedno ekstremum [5] (rys. 8).



Rys. 8. Przykłady empirycznej dekompozycji modów funkcji; na rysunku znajdują się otrzymane funkcje IMF oraz końcowe residua z dwóch różnych sygnałów wejściowych [9]

3. Baza danych

W celu przeprowadzenia testów skuteczności zaimplementowanego algorytmu wykorzystano dwie bazy danych stworzone przez Zbigniewa Kurka i Annę Tarnawską w ramach prac inżynierskich. Próbki zostały nagrane w budynku Politechniki Wrocławskiej przy pomocy sprzętu:

- Mikrofon Shure SM 58,
- Mikser Behringer: Eurorack MX802A,
- Zewnętrzny interfejs audio Creative Soundblaster Audigy 2 NX,
- Komputer PC.

Próbki zostały nagrane w formacie PCM Signed i zapisane w pliku w formacie wav z częstotliwością próbkowania 44100 Hz i 16-bitową głębią.

3.1. Baza pierwsza

Nagranie sześciu par będących w stałym związku przez co najmniej okres roku. Przy każdej sesji nagraniowej brała udział jedna para. Wcześniej zostały przygotowane zestawy pytań, odpowiednio jedna wersja dla kobiet, druga dla mężczyzn. Pytania były tak sformułowane, żeby odpowiedzi były słowami „tak” lub „nie”. Jedna osoba z pary zadawała pytanie, a odpowiedź drugiej była nagrywana mikrofonem. Istotnym elementem było opracowanie pytań, które miały wywołać stres u osoby odpowiadającej. Dlatego pytania zostały podzielone na trzy kategorie:

- pytania istotne: dotyczyły relacji między partnerami, główny temat „przesłuchania”, pytania mające na celu wzbudzenie stresu u osoby odpowiadającej;

- pytania nieistotne: pytania niezwiązane z tematem stanowiły przerywnik między pytaniami istotnymi;

- pytania kontrolne: osoba nagrywana miała na nie celowo skłamać.

Po zakończeniu badania każda z osób indywidualnie i anonimowo odpowiedziała jeszcze raz na pytania w celu zweryfikowania prawdomówności w trakcie nagrania [6].

3.2. Baza druga

Nagranie ośmiu osób — trójki kobiet i pięciu mężczyzn w różnym wieku. Podobnie jak w poprzedniej bazie pytania były zadawane w takiej postaci, żeby odpowiedź brzmiała „tak” lub „nie”. Nagranie było przeprowadzane w towarzystwie dwóch osób: zadającej pytanie i odpowiadającej na nie. Pytania poruszały kwestie, co do których ludzie byliby skłonni skłamać. Analogicznie do bazy pierwszej użyto trzech pytań: istotnych, nieistotnych i kontrolnych. Po skończeniu nagrań pytania skonsultowano z każdą osobą na osobności w celu weryfikacji odpowiedzi [11].

4. Metodyka

4.1. Założenia projektowe aplikacji

Podczas pisania aplikacji zostały przyjęte następujące założenia projektowe:

- Wykrycie stresu odbywa się tylko na podstawie wydobytego z próbki mikrodrżenia.

Program będzie określał zawartość stresu tylko na podstawie podstawowych założeń techniki VSA. Nie będzie korzystał z innych sposobów umożliwiających weryfikację stresu. Pozwoli to zbadać skuteczność zaimplementowanego algorytmu bez wpływu czynników zewnętrznych.

- Jednoznaczne określenie czy stres występuje w badanej próbce dźwiękowej.

Program nie powinien pozostawiać decyzji o występowaniu stresu użytkownikowi. Eliminuje to problem zróżnicowanego wyszkolenia osoby przeprowadzającej badanie. Dodatkowo umożliwia to jednoznaczne określenie skuteczności zaimplementowanej procedury.

- Obsługa różnych sygnałów audio.

Program powinien obsługiwać różne formaty i kodowania sygnału audio. Osoba przeprowadzająca badanie może posiadać próbki dźwiękowe nagrane w różnych formatach. Uciążliwe byłoby konwertowanie każdego pliku, dlatego program powinien posiadać możliwość zrobienia tego samemu przed dokonaniem analizy. Sposób w jaki sygnał został zakodowany nie powinien wpływać na przebieg detekcji stresu w nagraniu.

- Niezależność od systemu operacyjnego.

Program powinien być w miarę możliwości niezależny od systemu operacyjnego. Przenośność programu na różne platformy komputerowe daje dużą swobodę użytkownikowi i brak potrzeby posiadania konkretnego sprzętu do korzystania z aplikacji.

- Zapisanie raportu z dokonanych analiz.

Program powinien oferować możliwość zapisania wyników przeprowadzonych analiz podczas użytkowania aplikacji, co ułatwia późniejszą pracę statystyczną z otrzymanymi wynikami.

- Nagrywanie dźwięku.

Program powinien umożliwiać nagranie dźwięku przy pomocy mikrofonu, zapisanie go, oraz poddanie analizie pod kątem wystąpienia mikrodrżenia. Dodatkowo program powinien posiadać możliwość wyboru sygnału wejściowego.

- Odtwarzanie dźwięku.

Program powinien posiadać możliwość odtworzenia dźwięku, w celu weryfikacji nagrania albo wczytanego pliku dźwiękowego.

- Wygodny interfejs użytkownika.

Program powinien posiadać wygodny, intuicyjny interfejs użytkownika. Obsługa nie powinna sprawiać trudności, ani wymagać dużego nakładu czasowego na naukę użytkowania. Interfejs nie powinien pozwalać na dokonanie niedozwolonych operacji w programie.

- Graficzna prezentacja wyników.

Wyniki powinny być prezentowane w sposób graficzny, dające lepszy przegląd tego, co dzieje się w programie. Pozwala to zweryfikować poprawność podjętej decyzji przez program.

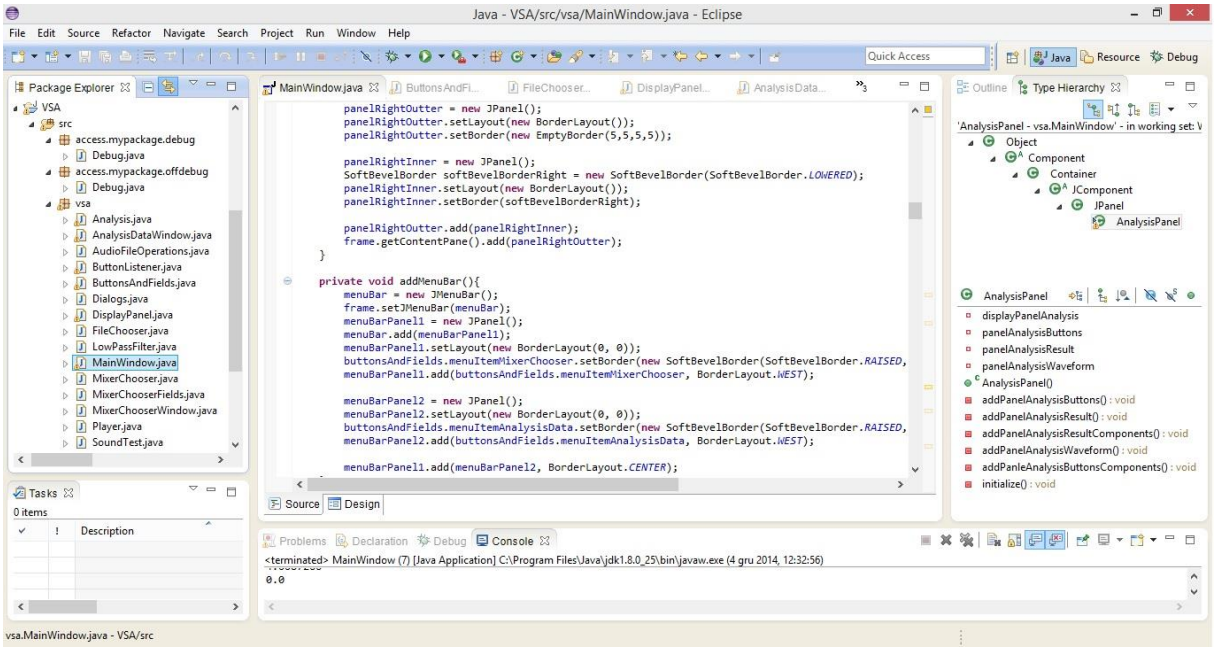
4.2. Język programowania i środowisko programistyczne

Do implementacji aplikacji VSA został wybrany język programowania Java. Głównymi motywami tej decyzji były:

- jest to język obiektowo zorientowany,
- duża ilość gotowych bibliotek, które są dobrze zoptymalizowane i skutecznie wspomagające pracę programisty,
- przenośność oprogramowania na różne platformy komputerowe,
- łatwe projektowanie interfejsu użytkownika,
- łatwe nasłuchiwanie i obsługa zdarzeń,
- przejrzysta składnia kodu.

4.2.1. Środowisko programistyczne Eclipse

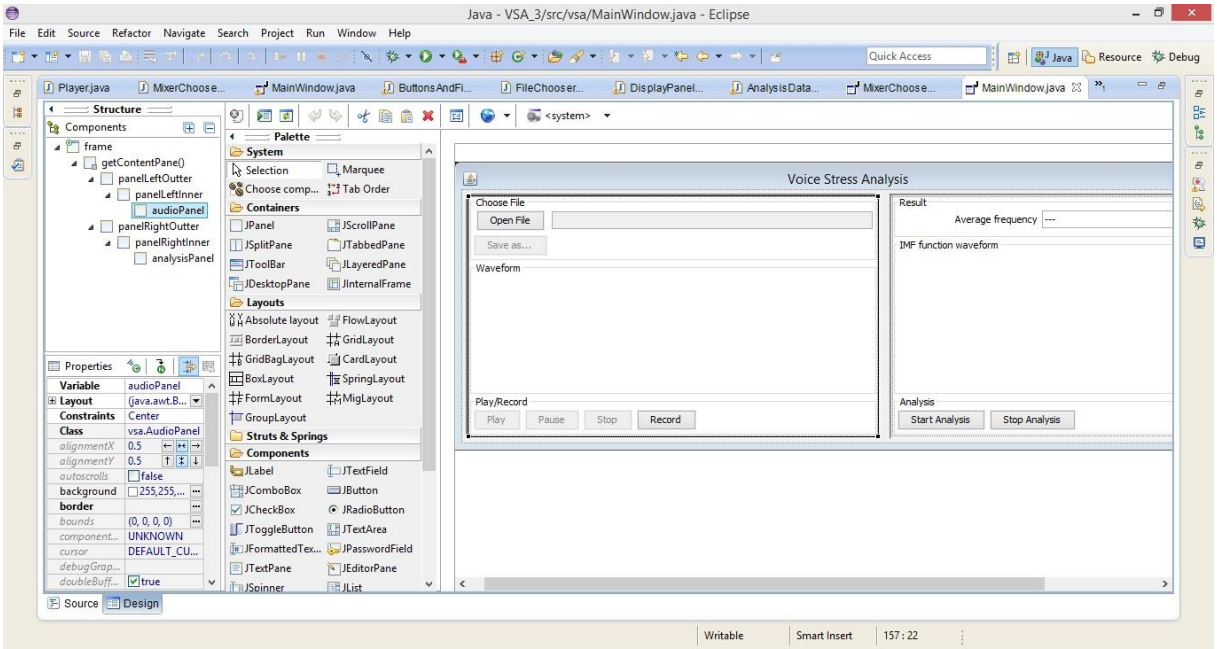
Do pisania aplikacji zostało wybrane darmowe środowisko programistyczne Eclipse Standard 4.3.1 z pakietu Eclipse Kepler SR1 (rys. 9). Jest to środowisko między innymi przeznaczone do pisania programów w Javie. Posiada wiele funkcji ułatwiających pracę nad programem. Jedną z nich jest rozwinięty edytor kodu programistycznego, podkreślający błędy i ostrzeżenia wykryte przez kompilator, oraz sugerujący możliwe rozwiązania; automatycznie importuje niezbędne pakiety; wyświetla podpowiedzi nazw obiektów i ich metod w trakcie pisania kodu; automatycznie porządkuje kod poprzez odpowiednie wcięcia i zaznacza różnymi kolorami różne części składni języka programowania. Kolejną funkcją jest możliwość korzystania z debuggera pozwalającego na krokową analizę programu, wliczając w to podgląd pracy różnych współbieżnych wątków, oraz wartości różnych zmiennych. Umożliwia przejrzenie hierarchii dziedziczenia przez dowolną klasę oraz sprawdzenie nazw implementowanych przez nią interfejsów, wyświetla także nazwy pól i metod oraz ich specyfikatory dostępu w wybranej klasie. Wszystkie te funkcje ułatwiają znalezienie i poprawienie błędów podczas pisania i testowania aplikacji.



Rys. 9. Widok okna środowiska programistycznego Eclipse

4.2.2. Dodatek WindowBuilder

Do środowiska został dodatkowo zainstalowany dodatek WindowBuilder (rys. 10) umożliwiający łatwą pracę z graficznym interfejsem użytkownika, działający na zasadzie przeciągania i upuszczania odpowiednich komponentów, oraz edycji ich wybranych parametrów. Praca z urządzeniami wspomagającymi pisanie zmniejsza czas potrzebny na stworzenie pożądanego wyglądu interfejsu użytkownika.



Rys. 10. Widok pracy w środowisku Eclipse z włączonym dodatkiem WindowBuilder

4.3. Wymagania sprzętowe

Wymagania sprzętowe umożliwiające uruchomienie aplikacji są uzależnione od tego, dla której wersji Wirtualnej Maszyny Javy JVM (Java Virtual Machine) zostaną przetłumaczone instrukcje kodu programu na kod bajtowy (Bytecode). Do kompilacji kodu został wykorzystany Pakiet Programisty Javy JDK (Java Development Kit) w wersji 8. W celu korzystania z aplikacji potrzebne jest zainstalowanie Środowiska Uruchomieniowego Javy JRE (Java Runtime Environment) w wersji 8, na które składają się Wirtualna Maszyna Javy i zestaw klas i narzędzi niezbędnych do uruchomienia programu napisanego w języku Java. Podsumowując, minimalnymi wymaganiami sprzętowymi dla działania aplikacji są minimalne wymagania oprogramowania Java 8.

4.3.1. Wymagania sprzętowe dla oprogramowania Java 8

1. System operacyjny
    - Windows 8 (Desktop)
    - Windows 7
    - Windows Vista SP2
    - Windows Server 2008 R2 SP1 (wersja 64-bitowa)
    - Windows Server 2012 (wersja 64-bitowa)
    - Mac OS X 10.8.3+
    - Mac OS X 10.9+
    - Oracle Linux 5.5+
    - Oracle Linux 6.x (wersja 32-bitowa), 6.x (wersja 64-bitowa)
    - Oracle Linux 7.x (wersja 64-bitowa)
    - Red Hat Enterprise Linux 5.5+, 6.x (wersja 32-bitowa), 6.x (wersja 64-bitowa)
    - Ubuntu Linux 12.04 LTS, 13.x
    - Suse Linux Enterprise Server 10 SP2+, 11.x

2. Procesor
    - Procesor Pentium 2 266 MHz lub lepszy

3. Miejsce na dysku twardym
    - 124 MB dla JRE
    - 2 MB dla funkcji Java Update

4. Pamięć RAM
    - 128 MB lub więcej [12]

4.4. Obsługiwane formaty audio

W programie została wykorzystana biblioteka *javax.sound.sampled* [13], która wspiera operacje na plikach audio. Aplikacja korzystając z tej biblioteki umożliwia wczytanie plików w formacie:

- WAVE,
- AIFC,
- AIFF,
- AU,
- SND.

Aplikacja umożliwia wczytanie plików audio kodowanych za pomocą:

- PCM Signed,
- PCM Unsigned;

z częstotliwością próbkowania:

- 8000 Hz,
- 11025 Hz,
- 16000 Hz,
- 22050 Hz,
- 32000 Hz,
- 44100 Hz,
- 48000 Hz,
- 96000 Hz;

z rozdzielczością bitową:

- 8 bitów,
- 16 bitów,
- 24 bity;

z liczbą kanałów:

- jeden kanał (mono),
- dwa kanały (stereo);

z kolejnością bajtów:

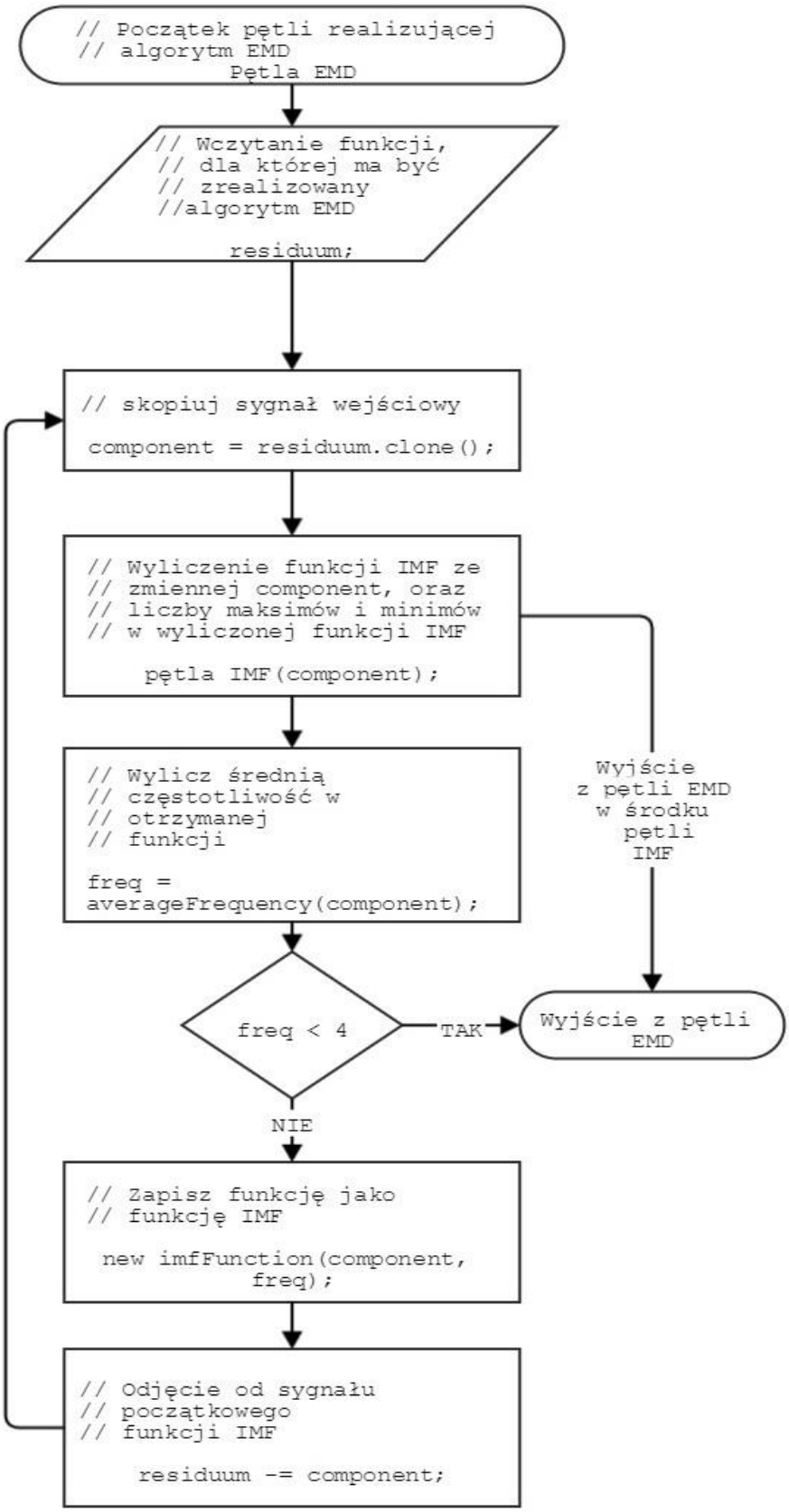
- little endian,
- big endian.

Aplikacja po wczytaniu pliku konwertuje go na format PCM Signed, 16 bitów, mono, little endian z częstotliwością próbkowania 44100 Hz oraz umożliwia nagrywanie w tym samym formacie.

4.5. Implementacja algorytmu

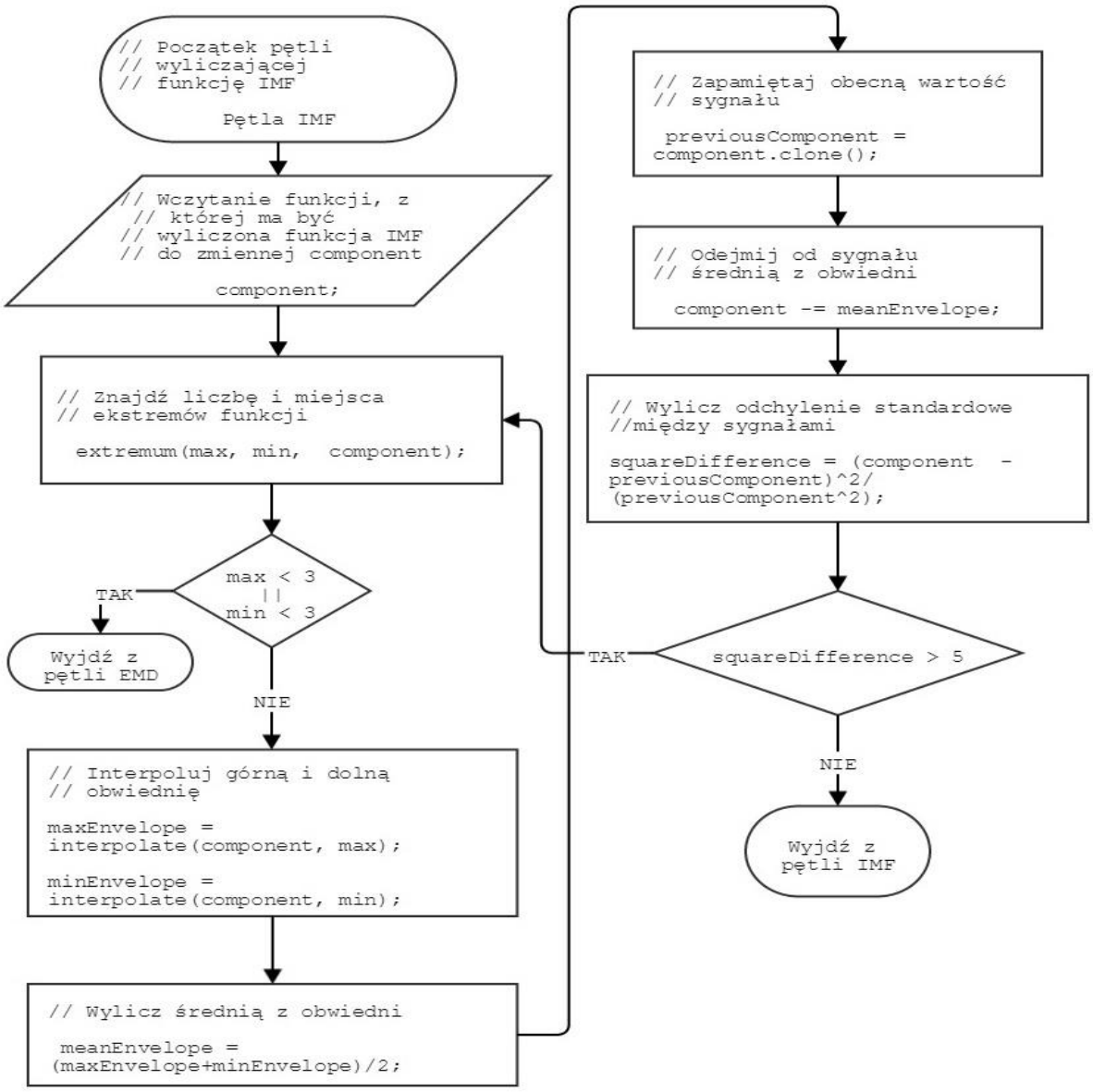
Najważniejszą częścią programu jest implementacja algorytmu EMD. Schemat blokowy metody realizującej procedurę EMD został podzielony dla zwiększenia czytelności na dwie części — pętlę zewnętrzną (rys. 11) wyliczającą kolejne residua funkcji wejściowej i pętlę wewnętrzną (rys. 12) wyszukującą funkcję IMF z podanego residuum.

Pętla zewnętrzna realizuje wyliczenie kolejnych residuum po sygnale wejściowym. Po znalezieniu funkcji IMF zostaje wyliczona jej częstotliwość. Jeżeli jest mniejsza niż 4 Hz algorytm kończy działania. Uzasadnione jest to faktem, że otrzymywanie jeszcze niższych częstotliwości nie ma wpływu na wykrycie mikrodrżenia w sygnale, które w przypadku braku stresu powinno się zawierać w przedziale 8–12 Hz. Jeżeli częstotliwość jest większa niż 4 Hz to jest zapamiętywana w celu późniejszego wybrania wartości najbliższej mikrodrżeniu. Otrzymana funkcja IMF jest odseparowywana od sygnału wejściowego.



Rys. 11. Pętla zewnętrzna procedury EMD

Wewnętrzna pętla realizuje wyliczanie kolejnych komponentów w celu znalezienia funkcji IMF. Najpierw są wyliczane lokalne ekstrema. Oba krańce przedziałów funkcji są traktowane jako jednocześnie maksimum i minimum, żeby możliwe było wyliczenie obwiedni za pomocą interpolacji. Jeżeli liczba maksimów albo minimów jest mniejsza od 3 algorytm jest przerywany. Uzasadnione jest to faktem, że jeżeli istnieją tylko dwa minima lub maksima, algorytm nie jest w stanie wyliczyć częstotliwości takiej funkcji. Kolejnym krokiem jest wyliczenie górnej i dolnej obwiedni funkcji na podstawie odnalezionych ekstremów. Następnie jest wyliczana średnia z obwiedni. Wartość komponentu jest chwilowo zapamiętywana, żeby było możliwe policzenie odchylenia standardowego. Średnia z obwiedni jest odejmowana od komponentu. Po wyliczeniu odchylenia standardowego między komponentem przed odjęciem i po odjęciu średniej z obwiedni następuje podjęcie decyzji czy otrzymana funkcja jest funkcją IMF. Wartość odchylenia standardowego, dla którego zmienia się decyzja, została dobrana empirycznie i wynosi 5. Jeżeli kryterium funkcji IMF nie jest spełnione, powtarza się pętla wewnętrzna w celu otrzymania kolejnego komponentu. Jeżeli otrzymywana funkcja spełnia kryteria, algorytm wraca do wykonywania pętli zewnętrznej.



Rys. 12. Pętla wewnętrzna procedury EMD

4.6. Opis najważniejszych klas i metod programu

Język programowania Java wywodzi się z założenia, że wszystko jest obiektem. Dlatego program został podzielony pod względem funkcjonalnym na różnego rodzaju klasy zawierające pola składowe opisujące stan obiektu i metody określające zachowanie i żądania jakie można na nim wykonywać [1]. W niniejszym podrozdziale zostały opisane najważniejsze klasy z punktu widzenia implementacji algorytmu EMD, ich ważniejsze pola i metody jakie mogą zostać wywołane na rzecz ich obiektów. Pełny wydruk kodu źródłowego tych klas znajduje się w Załączniku A.

4.6.1. Klasa AudioFileOperations

Klasa przeznaczona do różnych operacji na plikach audio, wliczając w to otwieranie i zapisywanie plików w odpowiednim formacie. Klasa, po otwarciu pliku audio, umożliwia otrzymanie zawartych w nim danych w postaci tablicy ciągu bajtów albo tablicy liczb zmiennoprzecinkowych pojedynczej precyzji. Pozwala na normalizację pliku do największej próbki oraz zawiera informacje o domyślnym formacie audio z jakim pracuje aplikacja.

Ważniejsze pola klasy

- `private static AudioFormat audioFormat;`

Zmienna przechowująca wykorzystywany w aplikacji format audio. Na klasę AudioFormat składają się pola określające kodowanie, częstotliwość próbkowania, głębię bitową, liczbę kanałów, rozmiar ramki w bajtach, liczbę ramek na sekundę i kolejność występowania bajtów w ramce.

Ważniejsze metody

- `public static float[] byteArrayIntoFloatArray(byte[]);`

Metoda dokonuje konwersji z otrzymanej po otwarciu pliku tablicy bajtów na tablicę zmiennych typu float. W programie są wykonywane na pliku audio metody operujące na typach zmiennoprzecinkowych (filtracja filtrem dolnoprzepustowy, interpolacja funkcjami sklejanymi trzeciego stopnia) Z sygnału wejściowego, w którym każda próbka reprezentowana jest przez 16 bitów (wartości od -32768 do 32767) otrzymywana jest tablica próbek reprezentowana przez wartości rzeczywiste z zakresu od -1 do 1. Parametrem metody jest tablica bajtów reprezentująca sygnał audio, a zwracana wartość to przekonwertowana tablica.

4.6.2. Klasa LowPassFilter

Klasa jest implementacją zaprojektowanego w programie Matlab filtru dolnoprzepustowego. Jest to filtr osiemsetnego rzędu o skończonej odpowiedzi impulsowej z oknem czasowym Blackmana. Częstotliwość odcięcia wynosi 400 Hz.

Ważniejsze pola klasy

- `private static final double[] coeff;`

Tablica współczynników filtru w postaci liczb zmiennoprzecinkowych podwójnej precyzji.

Ważniejsze metody

- `public static float[] filtering(float[]);`

Metoda przepuszczająca sygnał audio, reprezentowany przez tablicę liczb zmiennoprzecinkowych, przez filtr dolnoprzepustowy. Zwracaną wartością jest przefiltrowany sygnał. Wartość przefiltrowanej próbki jest wyliczana jako funkcja splotu n próbek sygnału z odwróconą kolejnością n współczynników filtru, gdzie n to liczba współczynników filtru. Przefiltrowanie sygnału ma na celu skrócenie czasu trwania algorytmu EMD poprzez wyeliminowanie składowych wysokoczęstotliwościowych, które są nieistotne dla wydobywanego mikrodrżenia.

4.6.3. Klasa IMFFunction

Klasa przechowująca wyliczoną w algorytmie EMD funkcję IMF. Dla każdej wyliczonej funkcji jest tworzony nowy obiekt klasy.

Ważniejsze pola klasy

- `float[] samples;`

Tablica typu float reprezentująca wyliczoną funkcję IMF.

- `float freq;`

Częstotliwość wyliczonej funkcji IMF.

4.6.4. Klasa MicrotremorFunction

Klasa przechowująca funkcję IMF będącą najbliżej przedziału mikrodrżenia decydującego o występowaniu bądź nie stresu.

Ważniejsze pola klasy

- `private boolean stressDetected;`

Zmienna typu logicznego określająca czy częstotliwość danej funkcji świadczy o występowaniu stresu.

4.6.5. Klasa Zeros

Klasa zawierająca liczbę oraz położenie miejsc zerowych

Ważniejsze pola klasy

- `int nrOfZeros;`

Liczba miejsc zerowych.

- `ArrayList<Integer> zeroPoints;`

Lista zawierająca położenie miejsc zerowych.

4.6.6. Klasa Analysis

Klasa wykonująca algorytm EMD. Istotne jest uwzględnienie w obliczeniach wykonywanych przez metody klasy występowania szumu kwantyzacji.

Ważniejsze pola klasy

- `private static final float delta;`

Przedział tolerancji przyjęty przy obliczeniach eliminujący szum kwantyzacji. Wartość tolerancji dalej nazywana ziarnem jest odpowiednikiem wartości jednego bita dla zmiennej szesnastobitowej w typie zmiennoprzecinkowy z zakresu -1 do 1.

Ważniejsze metody

- `private void extremum(ArrayList<Integer>, ArrayList<Integer>, float[]);`

Wyliczenie lokalnych maksimów i minimów w sygnale. Pierwsza i ostatnia próbka jest traktowana jako stan nieustalony i przyjmowana jednocześnie jako maksimum i minimum.

- `private float[] interpolation(float[], ArrayList<Integer>);`

Dokonuje interpolacji na podstawie położenia i wartości ekstremów.

- `private Zeros findingZeros(float[]);`

Znalezienie liczby i położenia miejsc zerowych. Metoda zlicza ile razy i w jakich miejscach wartość sygnału zmieniła znak o większą wartość niż połowa ziarna.

- `private float averageFrequency(float[]);`

Określenie częstotliwości na podstawie liczby przejść przez zero. Metoda do wyliczenia częstotliwości nie uwzględnia krańców przedziałów, które traktuje jako stany nieustalone. W celu poprawnego wykonania funkcja musi co najmniej dwa razy przechodzić przez miejsce zerowe nie licząc krańców przedziałów.

- `private void emd(float[], HashMap<Integer, IMFFunction>);`

Znalezienie kolejnych funkcji spełniających kryteria IMF w sygnale wejściowym. Metoda rozkłada sygnał na składowe o niższych częstotliwościach i jest implementacją algorytmu przedstawionego w podrozdziale 3.5. Po wyliczeniu każdej funkcji IMF jest określana jej częstotliwość i zapamiętywana w obiekcie klasy IMFFunction.

- `public MicrotremorFunction startAnalysis(float[]);`

Metoda rozpoczynająca analizę sygnału przekazywanego jako parametr. Metoda łączy działanie trzech innych. Przepuszcza sygnał przez filtr dolnoprzepustowy w celu ograniczenia składowych wysokoczęstotliwościowych. Znajduje kolejne funkcje IMF. Wylicza, która funkcja znajduje się najbliżej mikrodrżenia.

- `private MicrotremorFunction findClosestFrequencyToMicrotremor(HashMap<Integer, IMFFunction>);`

Metoda określa, która znaleziona funkcja IMF znajduje się najbliżej mikrodrżenia — częstotliwości 10 Hz. Następnie podejmowana jest decyzja na podstawie tego czy częstotliwość funkcji znajduje się w przedziale 8–12 Hz.

4.7. Czas działania algorytmu

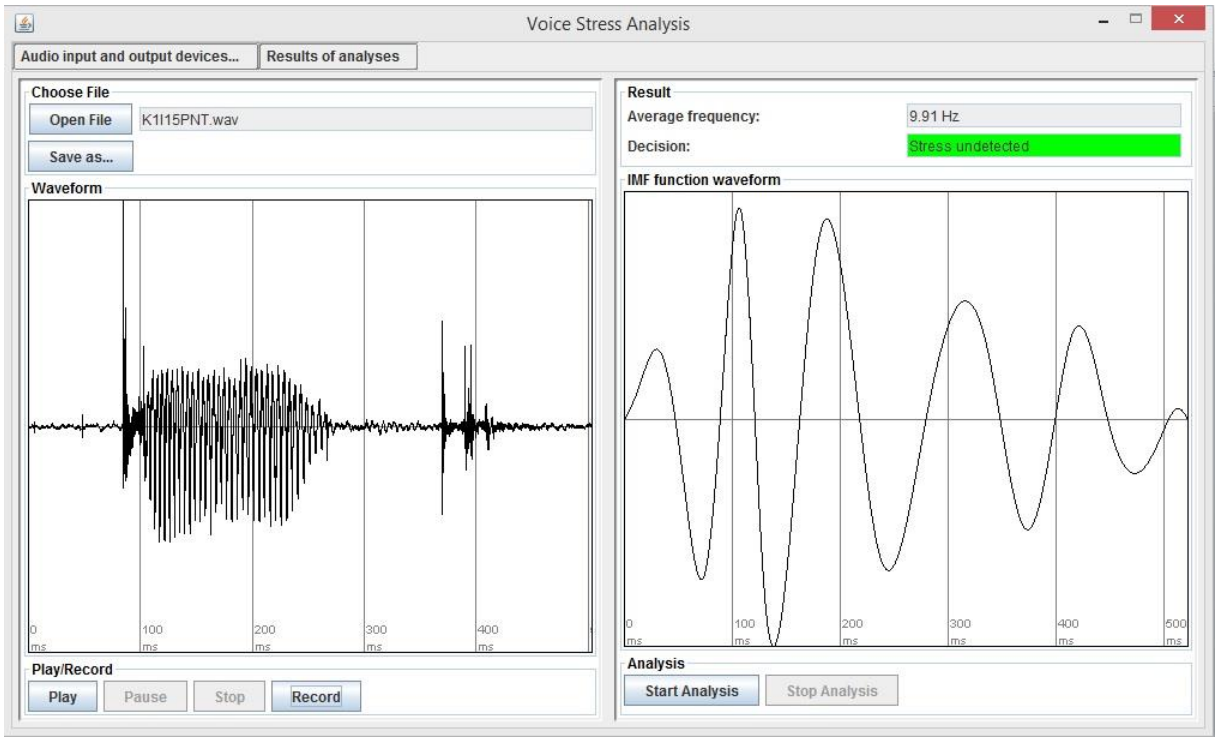
Czas wykonywania algorytmu jest uzależniony od czasu trwania nagrań dźwiękowych. W przypadku nagrań długości rzędu kilkuset milisekund program analizuje ich zawartość w zadowalającym czasie. Jednak w przypadku dłuższych czasów nagrań jest potrzebny większy nakład obliczeniowy, a co za tym idzie dłuższy czas wykonania algorytmu. Przy nagraniu mającym ponad jedną sekundę program potrzebuje kilku sekund na obliczenie częstotliwości w pobliżu mikrodrżenia.

5. Opis aplikacji

W aplikacji zaimplementowano opisany w rozdziale 3.5. algorytm. Po wczytaniu pliku audio, albo po nagraniu go, plik może zostać poddany analizie. Następnie podejmowana jest decyzja o występowaniu stresu w głosie i zostaje wyświetlona na ekranie głównym aplikacji. Dodatkowo jest prezentowana graficznie funkcja oraz jest wyświetlana jej częstotliwość, na podstawie której została podjęta decyzja. W aplikacji zostało przyjęte założenie, że jeżeli nie występuje sygnał składowy o częstotliwość mikrodrżenia z przedziału 8–12 Hz, dana osoba, której nagranie było poddane analizie, znajdowała się pod wpływem stresu. Historia otrzymywanych wyników jest zapamiętywana i umożliwione zostało zapisanie jej do pliku w formacie tekstowym.

Aplikacja została podzielona na segmenty pełniące określone funkcjonalności (rys. 13). Część po lewej stronie odpowiedzialna jest za pracę z wczytanym bądź nagranym plikiem audio. Część po prawej umożliwia rozpoczęcie analizy oraz prezentację otrzymanych rezultatów. Pasek menu zawiera dodatkowe możliwości aplikacji. Każda z części dzieli się na mniejsze segmenty, które zostały opisane w następnych podrozdziałach.

Kod źródłowy programu znajduje się w Załączniku B.



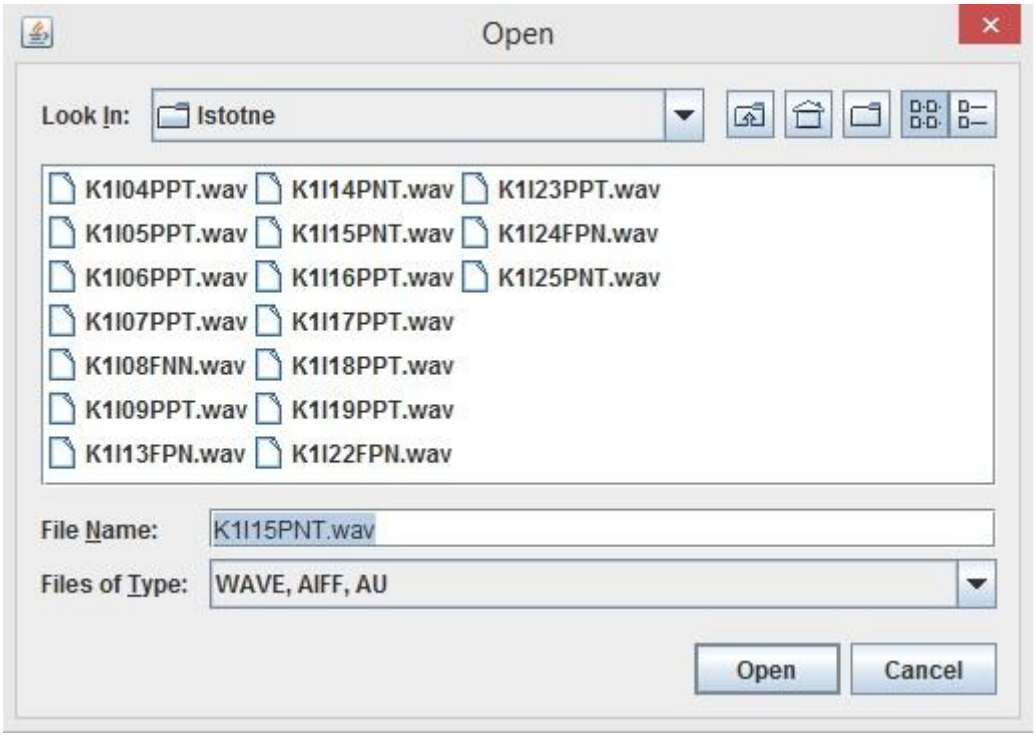
Rys. 13. Okno interfejsu użytkownika

5.1.Interfejs wyboru pliku

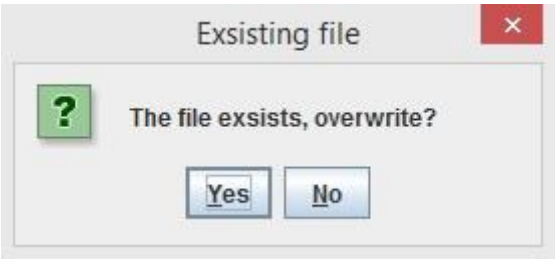
W lewym górnym rogu znajduje się interfejs umożliwiający wybór pliku do analizy (rys. 14). Umożliwia otworzenie pliku znajdującego się na dysku, zapisanie go pod inną nazwą oraz zapisanie nagranego sygnału przez aplikację. Po wczytaniu pliku jego skrócona nazwa jest prezentowana w polu tekstowym.

Rys. 14. Segment aplikacji odpowiedzialny za wybór pliku do analizy oraz zapis

Po wciśnięciu przycisku „Open File" pojawia się okno wyboru pliku (rys. 15), w którym został zaimplementowany filtr umożliwiający wybranie plików tylko z odpowiednim rozszerzeniem. Podobnie po wciśnięciu przycisku „Save File" pojawia się okno umożliwiające zapis pliku w formacie WAVE w wybranym katalogu. Jeżeli użytkownik nie wpisze nazwy pliku z rozszerzeniem, zostaje ono automatycznie dopisane do pliku. W przypadku próby nadpisania pliku pojawia się informujące o tym okno ostrzegawcze (rys. 16).



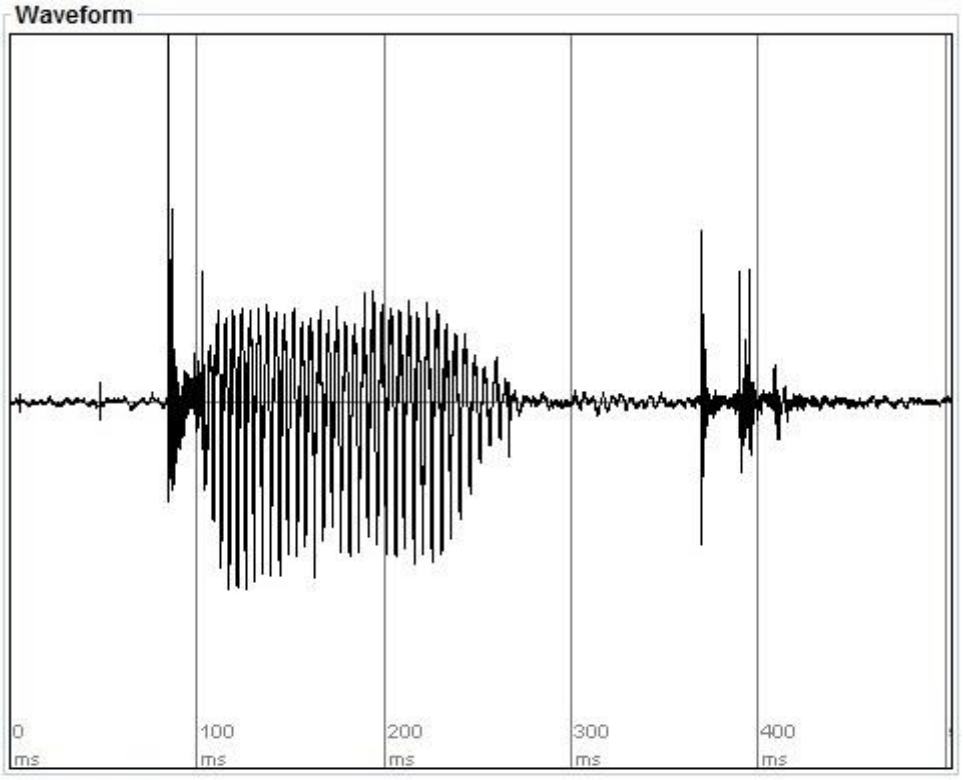
Rys. 15. Okno wyboru pliku do wczytania



Rys. 16. Okno informujące o próbie nadpisania pliku

5.2. Przedstawienie graficzne pliku przed analizą

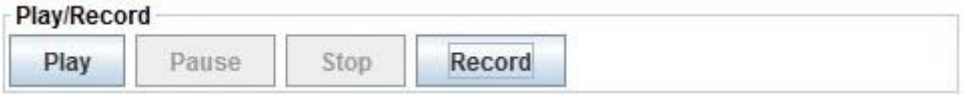
Aplikacja wyświetla podgląd przebiegu czasowego pliku dopasowując podziałkę w dziedzinie czasu (rys. 17). Wykres jest rysowany od razu po wczytaniu albo nagraniu pliku.



Rys. 17. Segment aplikacji przedstawiający przebieg czasowy pliku wczytanego do analizy

5.3. Interfejs odtwarzania i nagrywania

Program umożliwia odtworzenie wczytanego pliku, wstrzymanie, wznowienie i zatrzymanie odtwarzania (rys. 18). Dodatkowo można nagrać dźwięk i go odtworzyć. Umieszczenie tej funkcjonalności w aplikacji ma na celu umożliwienie weryfikacji słuchowej sygnału wejściowego.



Rys. 18. Część aplikacji umożliwiająca rozpoczęcie, wstrzymywanie, wznawianie i zatrzymywanie odtwarzania oraz nagrywanie

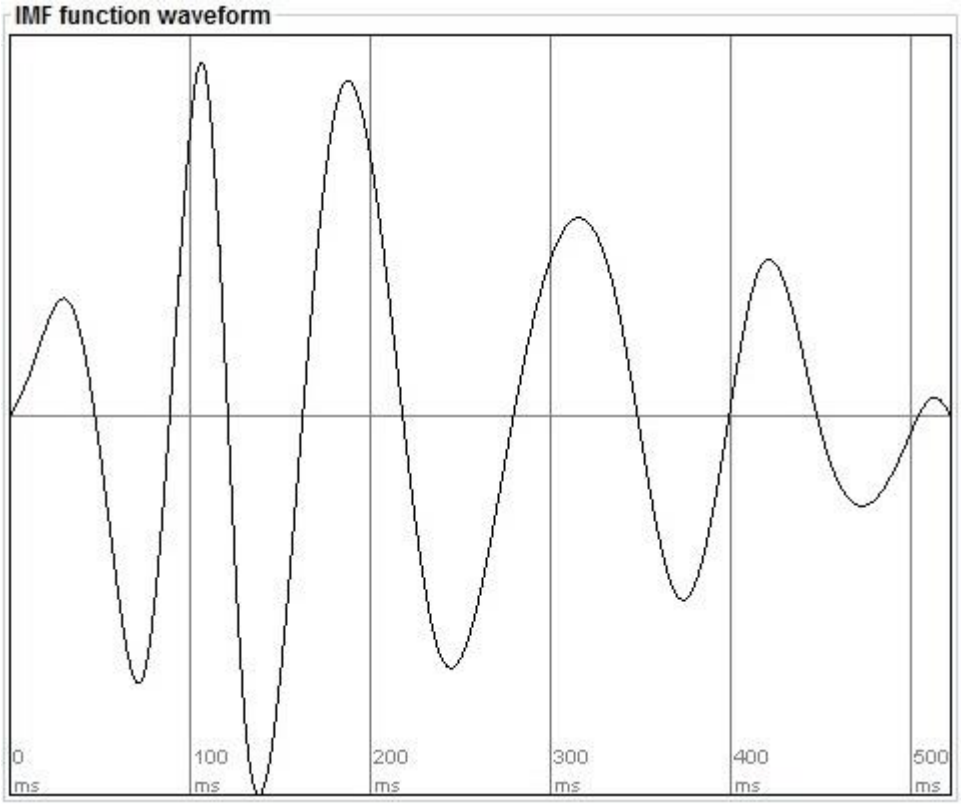
5.4. Interfejs rozpoczęcia i zatrzymania analizy

Po rozpoczęciu analizy aplikacja umożliwia przerwanie jej przez użytkownika (rys. 19). Jest to przydatne w przypadku próby poddania do analizy zbyt dużego pliku, co mogłoby okazać się czasochłonne. Przerwanie analizy jest wykonywane poprzez wygenerowanie specjalnego wyjątku w kodzie Javy, przechwycenie go i jego obsługę. Pozwala to na wykonanie wszystkich potrzebnych czynności w celu bezpiecznego przerwania obliczeń i wrócenia do stabilnego stanu programu.

Rys. 19. Segment aplikacji umożliwiający rozpoczęcie i przerwanie analizy

5.5. Przedstawienie graficzne funkcji odpowiadającej mikrodrżeniu

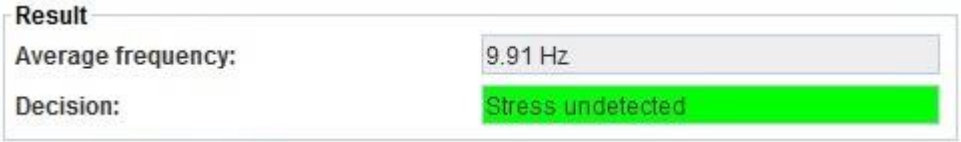
Aplikacja prezentuje w postaci graficznej mikrodrżenie wydobyte z sygnału (rys. 20). Jest to przebieg czasowy funkcji IMF położonej najbliżej częstotliwości 10 Hz. Podziałka na osi czasu jest dobierana automatycznie.



Rys. 20. Segment aplikacji rysujący przebieg czasowy funkcji IMF położonej najbliżej mikrodrżenia

5.6. Segment przedstawiający decyzję dotyczącą stresu w głosie

Najważniejszą częścią z punktu widzenia użytkowania programu jest segment prezentujący podjętą decyzję dotyczącą występowania stresu w głosie (rys. 21). W dolnym polu tekstowym jest wyświetlana decyzja i wyróżniana odpowiednim kolorem tła —zielonym dla niewykrycia stresu i czerwony dla wykrycia stresu w głosie. W górnym polu tekstowym jest wyświetlana częstotliwość funkcji IMF, na podstawie której została podjęta decyzja.



Rys. 21. Segment aplikacji prezentujący podjętą decyzję odnośnie występowania stresu

5.7. Pasek menu aplikacji

Na pasku menu aplikacji znajdują się dodatkowe funkcje programu.

- Wybór urządzenia wejściowego i wyjściowego w programie (rys. 22) — uruchamiany przyciskiem „Audio input and output devices…”

Aplikacja daje możliwość wybór urządzenia do nagrywania spośród dostępnych w komputerze, takich jak: wbudowany mikrofon, mikrofon podłączony do wejścia liniowego itp. Urządzenia są opisywane za pomocą odpowiednich „mikserów” systemowych komunikujących się z zewnętrznymi peryferiami [13]. Interfejs umożliwia sprawdzenie działania wybranego „miksera” poprzez próbę nagrania i odtworzenia dźwięku.

Analogicznie jest możliwość wyboru urządzenia do odtwarzania dźwięku, np.: głośniki, słuchawki. Działanie wybranego „miksera” wyjściowego można sprawdzić poprzez odtworzenie kilkusekundowego pliku z szumem brązowym.

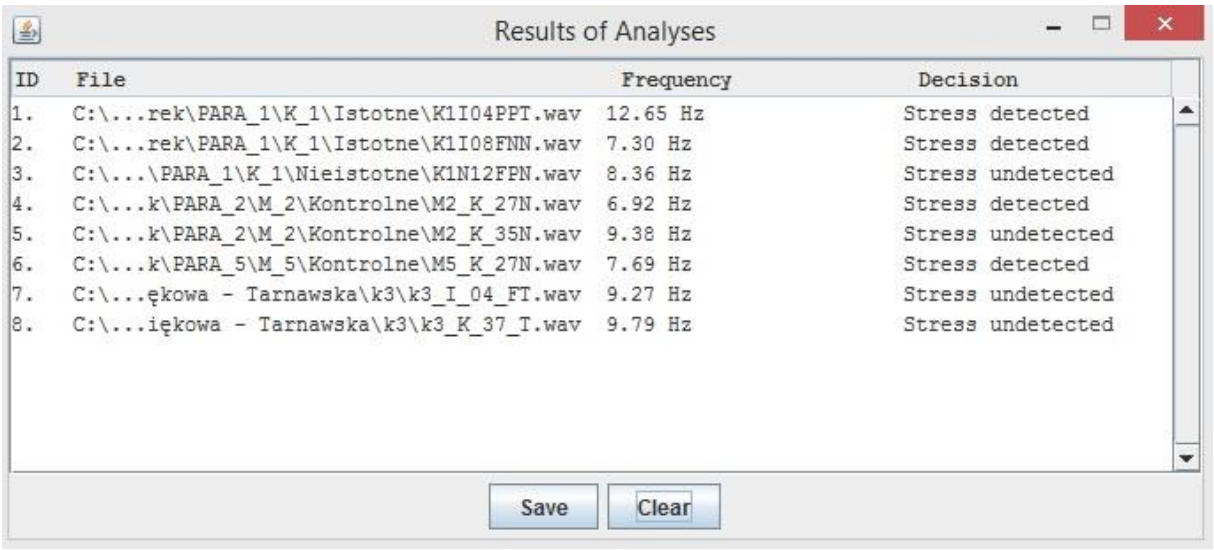
Wybór „mikserów” ma na celu umożliwienie ewentualnej pracy z zewnętrzną kartą dźwiękową. W przypadku korzystania tylko z funkcji analizy wcześniej nagranych próbek można wybór urządzeń wejściowych i wyjściowych pozostawić jako domyślny.

Warto zaznaczyć, że aplikacja nie włącza urządzenia (np. mikrofonu, głośników) jeżeli zostało odgórnie wyłączone przez system operacyjny. Dlatego w celu nagrywania dźwięku trzeba sprawdzić czy urządzenie zostało włączone w komputerze.

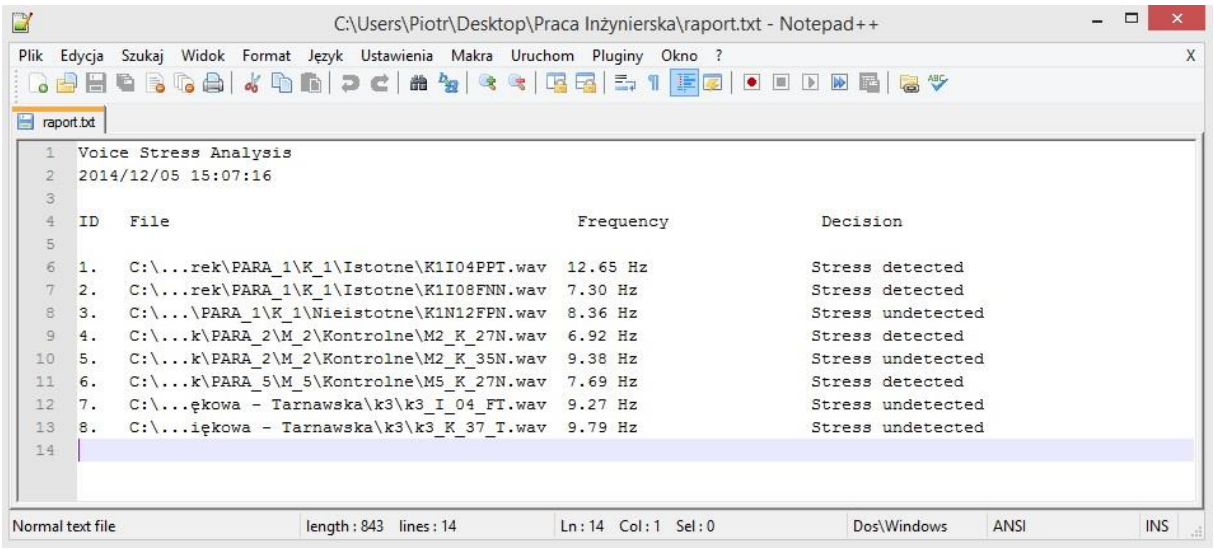
Rys. 22. Interfejs wyboru urządzenia wejściowego i wyjściowego aplikacji

- Historia przeanalizowanych plików dźwiękowych oraz możliwość zapisu wyników (rys. 23) — uruchamiana przyciskiem „Results of analyses”

Program zapamiętuje wyniki przeprowadzonych analiz plików i przedstawia je w postaci listy. Kolumny składają się kolejno z kolejności przeanalizowanej próbki (pole „ID”), skrótowego określenia ścieżki pliku (pole „File”), częstotliwości mikrodrżenia (pole „Frequency”) i podjętej decyzji (pole „Decision”). W celu wyczyszczenia historii należy wcisnąć przycisk „Clear”. Można wyświetloną listę wyników zapisać do pliku tekstowego w formacie „txt” (rys. 24). Pojawia się wtedy okno wyboru folderu i nazwy, tak jak w przypadku zapisu pliku dźwiękowego, oraz to samo okno ostrzegawcze, w przypadku próby nadpisania innego pliku. Plik tekstowy dodatkowo zawiera datę i godzinę dokonania zapisu.



Rys. 23. Okno aplikacji zawierające historię przeprowadzonych analiz



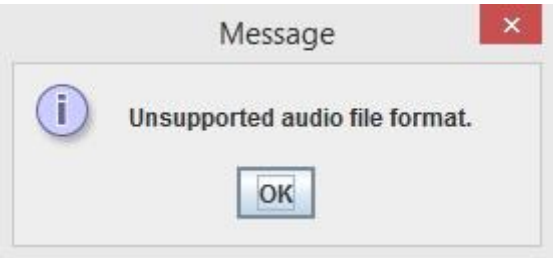
Rys. 24. Dokument tekstowy stworzony przez program

5.8. Pozostałe interfejsy aplikacji

Podczas zamykania aplikacji pojawia się okno potwierdzające operację mające na celu uniknięcia przypadkowego wyłączenia programu (rys. 25).

Rys. 25. Okno potwierdzenia wyjścia z aplikacji

Przy wystąpieniu błędów w trakcie działania programu użytkownik jest informowany wyskakującym oknem (rys. 26). Treść komunikatu jest uzależniona od przyczyny wystąpienia błędu.



Rys. 26. Wyskakujące okno w aplikacji informujące o wystąpieniu błędu

6. Testy skuteczności aplikacji

W celu sprawdzenia skuteczności algorytmu zostały przeprowadzone testy przy pomocy opisanej w rozdziale 4 bazy danych. Wyniki zostały porównane ze skutecznością algorytmów opracowanych w ramach prac Rutkowskiego [10] i Fleszara [2].

6.1. Metodyka badania skuteczności

Badanie skuteczności opiera się na wykonaniu zaimplementowanej w aplikacji procedury dla próbek z bazy. Aplikacja jednoznacznie podejmuje decyzję o wykryciu stresu. W trakcie badanie skuteczności jest konieczne przyjęcie założenie (takie jak podczas tworzenia bazy), że stres świadczy o kłamstwie, a brak stresu o mówieniu prawdy. Przy tworzeniu bazy, po nagraniach, była weryfikowana anonimowo prawdomówność, a co za tym idzie — stan emocjonalny, każdej z osób odpowiadających. Biorąc pod uwagę te dwa fakty możliwe jest porównanie wyników otrzymanych przy pomocy aplikacji z ich faktycznym stanem. Otrzymane wyniki działania algorytmu podzielono na cztery grupy.

- Prawda pozytywna (PP) — próbka w bazie danych została oznaczona jako nieprawdziwa i algorytm wskazał nieprawdę (wykrycie stresu)
- Prawda negatywna (PN) — próbka w bazie danych została oznaczona jako prawdziwa i algorytm wskazał prawdę (niewykrycie stresu)
- Fałsz pozytywny (FP) — próbka w bazie danych została oznaczona jako prawdziwa, a algorytm wskazał nieprawdę (wykrycie stresu)
- Fałsz negatywny (FN) — próbka w bazie danych została oznaczona jako nieprawdziwa, a algorytm wskazał prawdę (niewykrycie stresu)

Każda grupa składa się z dwuliterowego skrótu. Pierwsza litera określa czy algorytm poprawnie obliczył wynik (P – prawda, poprawnie, F – fałsz, niepoprawnie). Druga litera określa jaki wynik podał algorytm (P – pozytywny, wykrycie stresu, N – negatywny, niewykrycie stresu).

Skuteczność działania programu można przedstawić, jako procentowy stosunek liczby poprawnie wykrytych próbek do liczby wszystkich próbek poddanych analizie. Przedstawia się to wzorem:

$$Sk = \frac{PP+PN}{PP+PN+FP+FN} \cdot 100\% \, .$$

Do testów wzięte zostały tylko pytania istotne z baz danych, ponieważ w nich powinien znajdować się największy ładunek emocjonalny.

6.2. Wyniki testów skuteczności

W niniejszym podrozdziale przedstawiono wyniki testów dla pytań istotnych dla kolejnych osób w bazach oraz średnią skuteczność algorytmu dla płci, dla baz i dla algorytmu.

Tabela 1. Skuteczność działania zaimplementowanego algorytmu dla osób z bazy pierwszej

Osoba	PP [%]	PN [%]	FP [%]	FN [%]	**Skuteczność [%]**
Kobieta 1	5,9	47,1	29,4	17,6	**52,9**
Kobieta 2	5,9	47,0	41,2	5,9	**52,9**
Kobieta 3	17,6	35,4	29,4	17,6	**52,9**
Kobieta 4	11,8	53,0	17,6	17,6	**64,7**
Kobieta 5	0,0	47,1	29,4	23,5	**47,1**
Kobieta 6	5,9	47,1	29,4	17,6	**52,9**
Mężczyzna 1	17,6	53,0	17,6	11,8	**70,6**
Mężczyzna 2	5,9	47,1	23,5	23,5	**52,9**
Mężczyzna 3	11,8	41,2	23,5	23,5	**52,9**
Mężczyzna 4	11,8	58,8	11,8	17,6	**70,6**
Mężczyzna 5	5,9	64,7	11,8	17,6	**70,6**
Mężczyzna 6	17,6	41,3	23,5	17,6	**58,8**
Średnia	**9,8**	**48,6**	**24,0**	**17,6**	

Tabela 2. Skuteczność działania zaimplementowanego algorytmu dla osób z bazy drugiej

Osoba	PP [%]	PN [%]	FP [%]	FN [%]	**Skuteczność [%]**
Kobieta 1	13,0	30,4	0,0	56,6	**43,4**
Kobieta 2	17,4	52,2	8,7	21,7	**69,6**
Kobieta 3	0,0	27,3	27,3	45,5	**27,3**
Mężczyzna 1	17,4	52,2	13,0	17,4	**69,6**
Mężczyzna 2	17,4	43,5	13,0	26,1	**60,9**
Mężczyzna 3	21,7	39,2	13,0	26,1	**60,9**
Mężczyzna 4	13,0	30,4	8,7	47,9	**43,5**
Mężczyzna 5	13,0	34,9	21,7	30,4	**47,8**
Średnia	**14,1**	**38,8**	**13,2**	**34,0**	

Tabela 3. Skuteczność działania zaimplementowanego algorytmu dla płci w bazach, skuteczność dla całej bazy oraz ogólna skuteczność algorytmu

Baza i płeć	Średnia skuteczność dla płci [%]	Średnia skuteczność dla bazy [%]	**Średnia skuteczność algorytmu [%]**
Baza 1 kobiety	53,9	58,3	**56,1**
Baza 1 mężczyźni	62,7		
Baza 2 kobiety	46,8	52,9	
Baza 2 mężczyźni	56,5		

6.3. Analiza i porównanie testów skuteczności

Średnia skuteczność zaimplementowanego algorytmu w przeprowadzonym teście wynosi 56,1 %. Dla niektórych osób skuteczność przekraczała 70 %, co jest optymistycznym wynikiem. W obu bazach test wykonany na nagraniach mężczyzn przyniósł większą skuteczność, co może świadczyć o większej prostoliniowości i trudności ukrywania emocji u mężczyzn.

Warto zauważyć, że przyjęta do testów baza nie była zadowalająca, co mogło wpłynąć na uzyskaną przez algorytm skuteczność. Uzasadnione to jest następującymi przyczynami:

- Podczas przeprowadzania testu zostało przyjęte założenie, że osoba mówiąca kłamstwo znajduje się pod wpływem stresu, a osoba mówiąca prawdę nie znajduje się pod wpływem stresu. Zależność ta z oczywistych powodów nie jest idealnym założeniem. Osoba może się stresować niezależnie od udzielanej odpowiedzi. W szczególności przyznając się do czegoś, co chciałoby się zataić też działa się pod wpływem stresu mimo mówienia prawdy. I odwrotna sytuacja, można coś zataić, nie czując przy tym uczucia stresu.

- Dobór próbek bazy, na których zostało zrealizowane nagranie. Osoby, które znajdowały się w bazie danych mogły traktować tworzenie bazy, jako zwykłą zabawę. Mogły skłamać, bądź odpowiedzieć prawdziwie. Nie towarzyszyły ich odpowiedzią żadne konsekwencje w przeciwieństwie do przypadku, w którym jest realizowane prawdziwe przesłuchanie świadka albo winnego. W przypadku prawdziwego przesłuchania zawartość emocjonalna odpowiedzi jest z pewnością zdecydowanie większa. W przypadku braku żadnej odpowiedzialności powiązanie stresu z kłamstwem staje się dużo trudniejsze.

- Konsultacja wyników po stworzeniu bazy. Nic nie obliguje osób, które brały udział w tworzeniu bazy danych, aby potem ponownie udzielić prawdziwej odpowiedzi na wcześniej zadane pytania. Jest to zależne tylko od ich zaangażowania w przeprowadzone badanie. Zakładając, że osoba skłamała podczas przeprowadzania badania, nic nie stoi na przeszkodzie, aby podczas konsultacji z osobą przeprowadzającą badanie ponownie skłamała.

Poniżej znajduje się zestawienie skuteczności zaimplementowanego algorytmu z algorytmami opracowanymi w ramach prac Rutkowskiego [10] (Oznaczone jako RVSAx) i Fleszara [2] (Oznaczone jako VSAF0 /…/). Dla zwiększenia czytelności algorytm zaimplementowany w ramach tej pracy został oznaczony jako LVSA.

Tabela 4. Zestawienie skuteczności zaimplementowanych algorytmów w różnych pracach, *wytłuszczonym drukiem została zaznaczona największa skuteczność dla bazy*

Algorytm	Średnia skuteczność dla bazy 1 [%]	Średnia skuteczność dla bazy 2 [%]
LVSA	58,3	52,9
RVSA1	54,2	52,7
RVSA2	57,6	54,9
RVSA3	60,9	55,6
RVSA4	59,2	53,1
VSAF0 z średnią arytmetyczną	**66,6**	54,2
VSAF0 z medianą	58,9	53,1
VSAF0 z sieciami neuronowymi	59,1	52,0
VSAF0 z sieciami Bayesa	60,5	**56,6**

Porównując otrzymane wyniki można zauważyć, że w ramach innych prac osiągnięto większą skuteczność większości algorytmów jednakże wyniki nie są mocno zbliżone do siebie i w większości przypadków różnią tylko o wartość do 2 %.

7. Wnioski

Wyniki przeprowadzonych badań mocno odbiegają od deklarowanej przez producentów skuteczności urządzeń VSA. Możliwe, że połączenie algorytmów przygotowanych w ramach różnych prac dyplomowych pozwoliłoby zwiększyć skuteczność działania detekcji stresu. Istotne dla weryfikacji skuteczności algorytmu byłoby zbadanie zależności pomiędzy kłamstwem, a uczuciem stresu u człowieka. Stres jest wywoływany przez wiele różnorodnych czynników takich jak uczucie zagrożenia, problemy rodzinne, finansowe, nadmiar pracy, brak czasu na wypoczynek, choroby, brak akceptacji czy zmiana środowiska. Trudnym wydaje się rozróżnienie tych czynników stresogennych od kłamstwa lub zatajenia informacji. Tymczasem osoby poddane eksperymentowi mogły się znajdować pod wpływem stresu lub nie, z powodu wielu innych czynników, niemających nic wspólnego z prawdomównością. Przeanalizowanie bazy pod względem stresu pozwoliłoby lepiej zweryfikować skuteczność zaimplementowanych algorytmów. Fakt osiągnięcia podobnych wyników skuteczności algorytmu detekcji stresu może świadczyć o źle zrealizowanych próbkach bazy danych. Dużo bardziej wiarygodna byłaby baza opracowana na podstawie nagrań osób będących na przesłuchaniu i od których zależałoby ewentualne stwierdzenie winy i poniesienie kary. Być może, oprócz niewiarygodnej próby uwzględnionej w badaniu, błędne jest również założenie, że mikrodrżenie jest skorelowane ze stresem. Należy jednak zwrócić uwagę na fakt, że procedura zaimplementowana przez Fleszara korzystała z zupełnie innych założeń teoretycznych i mimo to też odznaczała się podobną skutecznością. W przypadku tak wielu różnych możliwych i niezbadanych zmiennych nasuwa się jedynie wniosek, że równie skuteczną metodą detekcji kłamstwa mógłby okazać się tak zwany rzut monetą.

Jako metoda weryfikacji poprawności algorytmu mogłaby posłużyć wykonana na odpowiednia baza sztucznie zmodyfikowanych próbek dźwiękowych. Część z nich byłaby zmodulowana amplitudowo sygnałem około 10 Hz w celu pewności wystąpienia „sztucznego mikrodrżenia" w sygnale. Na drugą część zostałby nałożony filtr górnoprzepustowy skutecznie tłumiący częstotliwości mikrodrżenia. Tak przygotowana baza mogłaby posłużyć do zrobienia badań statystycznych poprawności wykrywania mikrodrżenia przez algorytm.

Po zwiększeniu skuteczności urządzenie korzystające z zaimplementowanej techniki VSA mogłoby stanowić narzędzie wspomagające pracę innych detektorów kłamstwa w kryminalistyce. Innym zastosowaniem mogłaby być wstępna selekcja osób stanowiących zagrożenie dla społeczeństwa w miejscach takich jak lotnisko, różne środki transportu publicznego, miejsca publiczne jak galerie handlowe, czy biurowce. Po wstępnej selekcji system mógłby zwracać większą uwagę na te osoby w celu upewnienia się, że wywołany u nich stres nie jest wynikiem podjętego planu wykonania czynności niezgodnych z prawem.

Wiele rzeczy pozostaje jeszcze niezbadanych i próby dalszego rozwijania technik VSA wydają się warte uwagi. Metoda posiada wiele zalet, a jedną z ważniejszych jest możliwość przeprowadzenia badania bez świadomości podejrzanego. Wydaje się to nie do przeceniania w przypadku badania osób wytrenowanych w zatajaniu prawdy. Przy korzystaniu z urządzeń wspomagających detekcję kłamstwa ważne jest uświadomienie sobie, że każde urządzenie i każdy algorytm jest omylny. W związku z tym, podczas procesu badania podejrzanych o kłamstwo, nie można się bezwzględnie kierować podjętą prze urządzenie decyzją. Owszem, wskazania techniczne powinny mieć znaczenie, jednak ostateczną decyzję o tym czy ktoś kłamie, czy też nie, podejmuje drugi człowiek. W związku z tym, przy tak trudnych decyzjach, nigdy nie może zabraknąć czysto ludzkiego, zdrowego rozsądku, którego żadna maszyna nie potrafi oszacować ani zastąpić.

Bibliografia

[1] Eckel B., *Thinking in Java. Edycja polska*, wydawnictwo Helion, 2006.

[2] Fleszar P., *Analiza zjawiska MMT (Micro Muscle Tremors) dla potrzeb VSA (Voice Stress Analysis) i detekcji kłamstwa w głosie*, Wrocław: Politechnika Wrocławska 2013.

[3] Haddag D., Walter S., Ratley R., Smith M., *Investigation and Evaluation of Voice Stress Analysis Technology*, 2002. [Online]. https://www.ncjrs.gov/pdffiles1/nij/193832.pdf (sprawdzono dnia 2014.12.02)

[4] Hopkins C. S., Benincasa D. S., Ratley R. J., Grieco J. J., *Evaluation of Voice Stress Analysis Technology*, Proceedings of 28th Hawaii International Conference on System Sciences, 2005.

[5] Huang N. E., *Introduction to the Hilbert-Huang Transform and Its Related Mathematical Problems*, [Online]. http://ts.zesoi.fer.hr/materijali/HuangHilbertTransform_5862_chap1.pdf (sprawdzono dnia 2014.12.02)

[6] Kurek Z., *Nagranie akustycznej bazy sygnałów dla potrzeb VSA (Voice Stress Analysis)*, Wrocław: Politechnika Wrocławska, 2010.

[7] Lippold O. C. J., *Oscillation in the Strech Reflex Arc and the Origin of the Rythmical, 8–12 c/s Component of Physiological Tremor*, The Jurnal of Psychology, 206, str. 359–382, 1970.

[8] Lippold O. C. J., Redfearn J. W. T., Vuco J., *The Rythmical Activity of Groups of Motor Units in the Voluntary Contraction of Muscle*, The Jurnal of Psychology, 137, str. 473–487, 1957.

[9] Rilling G., Flandrin P., Goncalves P., *On Empirical Mode Decomposition and Its Algorithms*, [Online]. http://perso.ens-lyon.fr/paulo.goncalves/pub/NSIP03_GRPFPG.pdf (sprawdzono dnia 2014.12.03)

[10] Rutkowski R., *Analiza skuteczności metod VSA (Voice Stress Analysis) dla detekcji kłamstwa w głosie*, Wrocław: Politechnika Wrocławska, 2012.

[11] Tarnawska A., *Przeprowadzenie nagrań i etykietyzacja akustycznej bazy stresu w głosie,* Wrocław: Politechnika Wrocławska, 2013.

[12] [Online] https://www.java.com/pl/download/help/sysreq.xml (sprawdzono dnia 2014.12.04).

[13] [Online] http://docs.oracle.com/javase/8/docs/api/javax/sound/sampled/package-summary.html (sprawdzono dnia 2014.12.04).

Spis ilustracji

Spis tabel

Załącznik A. Kod źródłowy programu ważniejszych klas ze względu na implementację algorytmu EMD

Załącznik A1. Kod źródłowy pliku AudioFileOperations.java

```
package vsa;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;

import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.UnsupportedAudioFileException;

import access.mypackage.offdebug.Debug;

/**********************************************************************************************
 * Klasa AudioFileOperations
 *
 * Klasa przeznaczona do różnych operacji na plikach audio wliczając w to
 * otwierania i zapisywania plików, konwersję z danych w postaci tablicy bajtów
 * na tablicę floatów czy też normalizację.
 *
 * ********************************************************************************************
 */
public class AudioFileOperations {

	/**-------------------------------------------------------------------------
	 * Pola prywatne klasy
	 *
	 * -------------------------------------------------------------------------
	 */

	private static AudioInputStream audioInputStream = null;
	private static final int BUFFER_SIZE = 8192;
	private static byte[] audioBytes;
	private static AudioFormat.Encoding encoding = AudioFormat.Encoding.PCM_SIGNED;
	private static float sampleRate = 44100.0F;
	private static int sampleSizeInBits = 16;
	private static int channels = 1;
	private static int frameSize = sampleSizeInBits / 8 * channels;
	private static float frameRate = 44100.0F;
	private static boolean isBigEndian = false;
	private static AudioFormat audioFormat = new AudioFormat(encoding, sampleRate,
sampleSizeInBits, channels, frameSize, frameRate, isBigEndian);
	private static final Object audioArrayLock = new Object();



	/**-------------------------------------------------------------------------
	 * Metody publiczne
	 *
	 * -------------------------------------------------------------------------
	 */


	/**
	 * Otworzenie strumienia z pliku o podanej ścieżce.
	 * Jeżeli wystąpią błędy audioInputStream jest ustawiony na null.
	 * @param filePath ścieżka do pliku
	 * @return zmienna typu boolean określająca czy udało się otworzyć plik
	 */
	public static boolean openFile(String filePath) {
		File file = new File(filePath);

		try {
```

```
                        AudioInputStream audioTempInputStream = AudioSystem.getAudioInputStream(file);

                        //Stworzenie strumienia z odpowiednim formatem audio
                        audioInputStream = AudioSystem.getAudioInputStream(audioFormat,
audioTempInputStream);

                } catch (UnsupportedAudioFileException e) {
                        Dialogs.showMessage("Unsupported audio file format.");
                        Debug.debug(e.toString());
                        audioInputStream = null;
                        return false;
                } catch (IOException e) {
                        Debug.debug(e.toString());
                        audioInputStream = null;
                        return false;
                }
                return convertStreamIntoByteArray();

        }

        /**
         * Zapisanie tablicy bajtów do pliku zawierających sygnał audio.
         * Jeżeli pojawią się błędy podczas zapisu, funkcja zwraca false.
         * @param tablica bajtów reprezentująca sygnał audio
         * @param ścieżka do pliku
         * @return zmienna typu boolean określająca czy operacja się powiodła
         */
        public static boolean saveFile(byte[] data, String path) {
                ByteArrayInputStream bais = new ByteArrayInputStream(data);
                AudioInputStream audioInputStreamToSave;
                audioInputStreamToSave = new AudioInputStream(bais, audioFormat, data.length /
audioFormat.getFrameSize());
                File file = new File(path);

                int numBytesToWrite = data.length;
                while(numBytesToWrite > 0) {
                        try {
                                numBytesToWrite -= AudioSystem.write(audioInputStreamToSave,
AudioFileFormat.Type.WAVE, file);
                        } catch (IOException e) {
                                Dialogs.showMessage("Unable to save a file");
                                return false;
                        }
                }
                return true;
        }

        /**
         * Konwersja tablicy bajtów na tablicę floatów z zakresu -1 do 1
         * @param bytesArray tablica bajtów reprezentująca sygnał audio
         * @return tablica floatów reprezentująca sygnał audio
         */
        public static float[] byteArrayIntoFloatArray(byte[] bytesArray) {

                //rozmiar sampla w bitach
                final int bitsPerSample = AudioFileOperations.getAudioFormat().getSampleSizeInBits();
                //rozmiar sampla w bajtach
                //każdy bajt ma 8 bitów
                final int bytesPerSample = bitsPerSample / 8;

                int numOfBytes = bytesArray.length;
                int numOfSamples = numOfBytes/bytesPerSample;
                long[] transfer = new long[numOfSamples];
                float[] samples = new float[numOfSamples];

                //Dla każdego sampla
                for(int i = 0, k = 0, b; i < numOfBytes; i+= bytesPerSample, k++) {
                        transfer[k] = 0L;
                        //Dla każdego bajtu w samplu
                        //Ponieważ kodowanie jest w little-endian to trzeba drugi bajt przesunąć o 8
bitów w lewo
                        //gdyby był 3ci bajt to o 16 bitów w lewo itd.
                        for(b = 0; b < bytesPerSample; b++)
                                transfer[k] |= ((bytesArray[i+b] & 0xffL) << (8*b));
                }
```

```
        //policzenie ile bajtów jest niezajętych przez sampel w zmiennej typu long
        final int signShift = 64 - bitsPerSample;

        //Dla każdego sampla przesuń go w lewo o ilość miejsc nie określających wartości
        //i w prawo o tą samą wartość (wypełnienie jedynkami z przodu wartości ujemnych)
        for(int i = 0; i < transfer.length; i++)
                transfer[i] = ((transfer[i] << signShift) >> signShift);

        //maksymalna wartość dodatnia sygnału dla danej głębi bitowej
        final long fullScale = (long)Math.pow(2.0, bitsPerSample - 1) - 1;


        //Podziel każdą wartość przez największą możliwą w danej głębi bitowej
        for(int i = 0; i < transfer.length; i++)
                samples[i] = (float)transfer[i] / (float)fullScale;

        return samples;
}

/**
 * Normalizacja do największej próbki
 */
public static void normalized(float[] samples) {
        float maxValue = 0;
        for(int i = 0; i < samples.length; i ++)
                if (maxValue < Math.abs(samples[i]))
                        maxValue = Math.abs(samples[i]);


        for(int i = 0; i < samples.length; i++)
                samples[i] = samples[i]/maxValue;
}

/**---------------------------------------------------------------------------
 *
 *      Gettery
 *
 *---------------------------------------------------------------------------
 */

/**
 * Otrzymanie tablicy bajtów reprezentujących sygnał audio
 * @return byte[] sygnał audio w bajtach
 */
public static byte[] getByteArrayWithAudio() {
        synchronized(audioArrayLock) {
                return audioBytes.clone();
        }
}

/**
 * Zwraca domyślny format audio używany w aplikacji.
 * @return AudioFormat
 */
public static AudioFormat getAudioFormat(){
        return audioFormat;
}



/**---------------------------------------------------------------------------
 * Metody prywatne
 *
 * ---------------------------------------------------------------------------
 */


/**
 * Konwersja strumienia wejściowego audio w tablicę bajtów.
 * @return zmienna typu boolean określająca powodzenie operacji
 */
private static boolean convertStreamIntoByteArray(){
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        byte[] data = new byte[BUFFER_SIZE];
        int numBytesRead = 0;
        try {
```

```
                        while((numBytesRead = audioInputStream.read(data)) != -1)
                                out.write(data, 0, numBytesRead);
                } catch (IOException e) {
                        Debug.debug(e.toString());
                        return false;
                }

                synchronized(audioArrayLock) {
                        audioBytes = new byte[out.size()];
                        audioBytes = out.toByteArray();
                }

                audioInputStream = null;

                try {
                        out.flush();
                        out.close();
                } catch (IOException e) {
                        Debug.debug(e.toString());
                        return false;
                }
                return true;
        }
}
```

Załącznik A2. Kod źródłowy pliku LowPassFilter.java

```
package vsa;

/*******************************************************************************
 * Klasa LowPassFilter
 *
 * Klasa realizująca filtr 800-nego rzędu o skończonej odpowidzi impulsowej.
 * Jest to filtr z oknem czasowym Blackmana o czętsotliwości odcięcia 400 Hz.
 *
 * ****************************************************************************
 */
public class LowPassFilter {


        /**
         * Współczynniki filtru
         */
        private static final double coeff[] = {
                0,
                -3.09236906250145e-09,
                -1.16621413750432e-08,
                -2.45557542337669e-08,
                -4.05046645954493e-08,
                -5.81366964073535e-08,
                -7.59883029998755e-08,
                -9.25176799418175e-08,
                -1.06118658751313e-07,
                -1.15135307104001e-07,
                -1.17877156703393e-07,
                -1.12634975798380e-07,
                -9.76969994672016e-08,
                -7.13655272565302e-08,
                -3.19737945852341e-08,
                2.20969784866114e-08,
                9.24004599459233e-08,
                1.80408100815630e-07,
                2.87492154892933e-07,
                4.14908926860107e-07,
                5.63782353123439e-07,
                7.35088020014686e-07,
                9.29637723804598e-07,
                1.14806467632067e-06,
                1.39080945882484e-06,
                1.65810682518491e-06,
                1.94997345326195e-06,
                2.26619674083339e-06,
```

2.60632473927529e-06,
2.96965731463963e-06,
3.35523862168689e-06,
3.76185097187243e-06,
4.18801017124830e-06,
4.63196239873433e-06,
5.09168268924768e-06,
5.56487507976972e-06,
6.04897446958912e-06,
6.54115023870543e-06,
7.03831166073072e-06,
7.53711513860731e-06,
8.03397328309023e-06,
8.52506584525360e-06,
9.00635250529271e-06,
9.47358751064512e-06,
9.92233614697004e-06,
1.03479930158466e-05,
1.07458020832089e-05,
1.11108784525704e-05,
1.14382318070432e-05,
1.17227914540664e-05,
1.19594328966690e-05,
1.21430058450537e-05,
1.22683635723364e-05,
1.23303935084684e-05,
1.23240489567487e-05,
1.22443818079481e-05,
1.20865761179763e-05,
1.18459824062601e-05,
1.15181525236349e-05,
1.10988749306199e-05,
1.05842102195066e-05,
9.97052670678619e-06,
9.25453591612900e-06,
8.43332776645153e-06,
7.50440527460958e-06,
6.46571857799615e-06,
5.31569807882518e-06,
4.05328650920915e-06,
2.67796971431396e-06,
1.18980594994375e-06,
-4.10546509099106e-07,
-2.12180365257879e-06,
-3.94203211108448e-06,
-5.86862782280389e-06,
-7.89829718465892e-06,
-1.00270408766353e-05,
-1.22501405420550e-05,
-1.45621484994031e-05,
-1.69568806531068e-05,
-1.94274127613892e-05,
-2.19660802089826e-05,
-2.45644814211058e-05,
-2.72134850426913e-05,
-2.99032409934363e-05,
-3.26231954948451e-05,
-3.53621101500907e-05,
-3.81080851412770e-05,
-4.08485865915686e-05,
-4.35704781217382e-05,
-4.62600566120110e-05,
-4.89030921607121e-05,
-5.14848722112379e-05,
-5.39902497983187e-05,
-5.64036958435200e-05,
-5.87093554085028e-05,
-6.08911077928367e-05,
-6.29326303412047e-05,
-6.48174658027601e-05,
-6.65290930632939e-05,
-6.80510010488354e-05,
-6.93667655774572e-05,
-7.04601289144841e-05,
-7.13150817651340e-05,
-7.19159474179411e-05,

-7.22474677322617e-05,
-7.22948906438355e-05,
-7.20440588438895e-05,
-7.14814992697408e-05,
-7.05945130283952e-05,
-6.93712653593427e-05,
-6.78008752287559e-05,
-6.58735041346801e-05,
-6.35804436916823e-05,
-6.09142015539089e-05,
-5.78685852276504e-05,
-5.44387833184556e-05,
-5.06214437536365e-05,
-4.64147485187423e-05,
-4.18184844463319e-05,
-3.68341095972090e-05,
-3.14648147782411e-05,
-2.57155797470351e-05,
-1.95932236621213e-05,
-1.31064493479167e-05,
-6.26588095666994e-06,
9.15905375230231e-07,
8.42435826167659e-06,
1.62429355573125e-05,
2.43530969473100e-05,
3.27343049554072e-05,
4.13640346626618e-05,
5.02177924007311e-05,
5.92691436527458e-05,
6.84897503620943e-05,
7.78494178147670e-05,
8.73161512243955e-05,
9.68562221108855e-05,
0.000106434244523717,
0.000116013261119668,
0.000125554839062057,
0.000135019175664793,
0.000144365213659611,
0.000153550765919208,
0.000162532649422522,
0.000171266828201537,
0.000179708564961776,
0.000187812581021308,
0.000195533224165903,
0.000202824643971039,
0.000209640974095084,
0.000215936521002345,
0.000221665958529974,
0.000226784527669252,
0.000231248240889671,
0.000235014090293815,
0.000238040258852403,
0.000240286333932417,
0.000241713522296960,
0.000242284865723817,
0.000241965456360702,
0.000240722650909088,
0.000238526282705587,
0.000235348870750147,
0.000231165824714202,
0.000225955644949308,
0.000219700116508094,
0.000212384496184519,
0.000203997691579672,
0.000194532431202796,
0.000183985424624896,
0.000172357511714387,
0.000159653800000676,
0.000145883789232598,
0.000131061482224025,
0.000115205481109003,
9.83390681632512e-05,
8.04902703878009e-05,
6.16919070940166e-05,
4.19816197769953e-05,
2.14018836163928e-05,

-1.87346740234822e-19,
-2.21719304743364e-05,
-4.50570549986815e-05,
-6.85938359691500e-05,
-9.27161391550566e-05,
-0.000117353342889120,
-0.000142430468466854,
-0.000167868331862513,
-0.000193583716785595,
-0.000219489569016230,
-0.000245495211870058,
-0.000271506582553813,
-0.000297426489082019,
-0.000323154887333398,
-0.000348589177733021,
-0.000373624520953468,
-0.000398154171935396,
-0.000422069831435609,
-0.000445262014219172,
-0.000467620432921791,
-0.000489034396519995,
-0.000509393222259934,
-0.000528586659811412,
-0.000546505326332225,
-0.000563041151049750,
-0.000578087827892006,
-0.000591541274629877,
-0.000603300096925844,
-0.000613266055623152,
-0.000621344535552874,
-0.000627445014085407,
-0.000631481527607764,
-0.000633373134068928,
-0.000633044369702842,
-0.000630425698012585,
-0.000625453949080161,
-0.000618072747254343,
-0.000608232925264461,
-0.000595892922810859,
-0.000581019167693471,
-0.000563586437558309,
-0.000543578200368100,
-0.000520986931737601,
-0.000495814407316512,
-0.000468071968453318,
-0.000437780759431793,
-0.000404971934638251,
-0.000369686834091835,
-0.000331977125852081,
-0.000291904913907528,
-0.000249542810245946,
-0.000204973969910915,
-0.000158292087960349,
-0.000109601357360110,
-5.90163869697242e-05,
-6.66207890693937e-06,
4.73265352867950e-05,
0.000102804500117456,
0.000159617183538490,
0.000217600566422777,
0.000276581574096398,
0.000336378449767734,
0.000396801169526488,
0.000457651898426015,
0.000518725486999747,
0.000579810007398850,
0.000640687328174275,
0.000701133726562733,
0.000760920536973294,
0.000819814834210144,
0.000877580149807949,
0.000933977219700128,
0.000988764761287588,
0.00104170027782691,
0.00109254088791312,
0.00114104417769388,

0.00118696907331928,
0.00123007673100590,
0.00127013144197516,
0.00130690154941507,
0.00134016037451232,
0.00136968714850789,
0.00139526794764541,
0.00141669662780701,
0.00143377575556755,
0.00144631753234498,
0.00145414470828206,
0.00145709148246451,
0.00145500438606125,
0.00144774314496587,
0.00143518151852358,
0.00141720811094582,
0.00139372715204505,
0.00136465924396516,
0.00132994207063897,
0.00128953106677230,
0.00124340004323567,
0.00119154176583799,
0.00113396848456284,
0.00107071241046631,
0.00100182613756534,
0.000927383007187891,
0.000847477412409063,
0.000762225040361889,
0.000671763050386084,
0.000576250186162794,
0.000475866820177766,
0.000370814929058448,
0.000261317998542189,
0.000147620857051816,
2.99894370812383e-05,
-9.12895361739023e-05,
-0.000215908929267415,
-0.000343541658119669,
-0.000473841218638007,
-0.000606442289928120,
-0.000740961409431809,
-0.000876997719073311,
-0.00101413378124239,
-0.00115193646318889,
-0.00128995788815139,
-0.00142773645129277,
-0.00156479789826871,
-0.00170065646401258,
-0.00183481606908293,
-0.00196677157068766,
-0.00209601006527473,
-0.00222201223936171,
-0.00234425376506855,
-0.00246220673661824,
-0.00257534114388189,
-0.00268312637886651,
-0.00278503277087791,
-0.00288053314593791,
-0.00296910440589436,
-0.00305022912253643,
-0.00312339714191533,
-0.00318810719397393,
-0.00324386850250701,
-0.00329020239040843,
-0.00332664387511245,
-0.00335274324910371,
-0.00336806764035525,
-0.00337220254755524,
-0.00336475334500267,
-0.00334534675208861,
-0.00331363226233381,
-0.00326928352702531,
-0.00321199968858364,
-0.00314150665889885,
-0.00305755833799688,
-0.00295993776853854,

-0.00284845822180985,
-0.00272296421103557,
-0.00258333242803659,
-0.00242947259945513,
-0.00226132825899035,
-0.00207887743231902,
-0.00188213323162117,
-0.00167114435688899,
-0.00144599550146622,
-0.00120680765954653,
-0.000953738333649181,
-0.000686981640390124,
-0.000406768313174500,
-0.000113365600750776,
0.000192922939111974,
0.000511757751236587,
0.000842763700023204,
0.00118553059155805,
0.00153961380026185,
0.00190453499915929,
0.00227978299251661,
0.00266481464926198,
0.00305905593527354,
0.00346190304229443,
0.00387272361091363,
0.00429085804473733,
0.00471562091256858,
0.00514630243511501,
0.00558217005245541,
0.00602247006821795,
0.00646642936615642,
0.00691325719455678,
0.00736214701366593,
0.00781227840110885,
0.00826281901004885,
0.00871292657465180,
0.00916175095723639,
0.00960843623133246,
0.0100521227947268,
0.0104919495064521,
0.0109270558415695,
0.0113565840575113,
0.0117796813656839,
0.0121955021019863,
0.0126032098898760,
0.0130019797896094,
0.0133910004272999,
0.0137694760974767,
0.0141366288328835,
0.0144917004353342,
0.0148339544615464,
0.0151626781579853,
0.0154771843388969,
0.0157768132018614,
0.0160609340753778,
0.0163289470931856,
0.0165802847902395,
0.0168144136154873,
0.0170308353568409,
0.0172290884739977,
0.0174087493350405,
0.0175694333530357,
0.0177107960191520,
0.0178325338291326,
0.0179343851002827,
0.0180161306764628,
0.0180775945189253,
0.0181186441811778,
0.0181391911664141,
0.0181391911664141,
0.0181186441811778,
0.0180775945189253,
0.0180161306764628,
0.0179343851002827,
0.0178325338291326,
0.0177107960191520,

0.0175694333530357,
0.0174087493350405,
0.0172290884739977,
0.0170308353568409,
0.0168144136154873,
0.0165802847902395,
0.0163289470931856,
0.0160609340753778,
0.0157768132018614,
0.0154771843388969,
0.0151626781579853,
0.0148339544615464,
0.0144917004353342,
0.0141366288328835,
0.0137694760974767,
0.0133910004272999,
0.0130019797896094,
0.0126032098898760,
0.0121955021019863,
0.0117796813656839,
0.0113565840575113,
0.0109270558415695,
0.0104919495064521,
0.0100521227947268,
0.00960843623133246,
0.00916175095723639,
0.00871292657465180,
0.00826281901004885,
0.00781227840110885,
0.00736214701366593,
0.00691325719455678,
0.00646642936615642,
0.00602247006821795,
0.00558217005245541,
0.00514630243511501,
0.00471562091256858,
0.00429085804473733,
0.00387272361091363,
0.00346190304229443,
0.00305905593527354,
0.00266481464926198,
0.00227978299251661,
0.00190453499915929,
0.00153961380026185,
0.00118553059155805,
0.000842763700023204,
0.000511757751236587,
0.000192922939111974,
-0.000113365600750776,
-0.000406768313174500,
-0.000686981640390124,
-0.000953738333649181,
-0.00120680765954653,
-0.00144599550146622,
-0.00167114435688899,
-0.00188213323162117,
-0.00207887743231902,
-0.00226132825899035,
-0.00242947259945513,
-0.00258333242803659,
-0.00272296421103557,
-0.00284845822180985,
-0.00295993776853854,
-0.00305755833799688,
-0.00314150665889885,
-0.00321199968858364,
-0.00326928352702531,
-0.00331363226233381,
-0.00334534675208861,
-0.00336475334500267,
-0.00337220254755524,
-0.00336806764035525,
-0.00335274324910371,
-0.00332664387511245,
-0.00329020239040843,
-0.00324386850250701,

-0.00318810719397393,
-0.00312339714191533,
-0.00305022912253643,
-0.00296910440589436,
-0.00288053314593791,
-0.00278503277087791,
-0.00268312637886651,
-0.00257534114388189,
-0.00246220673661824,
-0.00234425376506855,
-0.00222201223936171,
-0.00209601006527473,
-0.00196677157068766,
-0.00183481606908293,
-0.00170065646401258,
-0.00156479789826871,
-0.00142773645129277,
-0.00128995788815139,
-0.00115193646318889,
-0.00101413378124239,
-0.000876997719073311,
-0.000740961409431809,
-0.000606442289928120,
-0.000473841218638007,
-0.000343541658119669,
-0.000215908929267415,
-9.12895361739023e-05,
2.99894370812383e-05,
0.000147620857051816,
0.000261317998542189,
0.000370814929058448,
0.000475866820177766,
0.000576250186162794,
0.000671763050386084,
0.000762225040361889,
0.000847477412409063,
0.000927383007187891,
0.00100182613756534,
0.00107071241046631,
0.00113396848456284,
0.00119154176583799,
0.00124340004323567,
0.00128953106677230,
0.00132994207063897,
0.00136465924396516,
0.00139372715204505,
0.00141720811094582,
0.00143518151852358,
0.00144774314496587,
0.00145500438606125,
0.00145709148246451,
0.00145414470828206,
0.00144631753234498,
0.00143377575556755,
0.00141669662780701,
0.00139526794764541,
0.00136968714850789,
0.00134016037451232,
0.00130690154941507,
0.00127013144197516,
0.00123007673100590,
0.00118696907331928,
0.00114104417769388,
0.00109254088791312,
0.00104170027782691,
0.000988764761287588,
0.000933977219700128,
0.000877580149807949,
0.000819814834210144,
0.000760920536973294,
0.000701133726562733,
0.000640687328174275,
0.000579810007398850,
0.000518725486999747,
0.000457651898426015,
0.000396801169526488,

0.000336378449767734,
0.000276581574096398,
0.000217600566422777,
0.000159617183538490,
0.000102804500117456,
4.73265352867950e-05,
-6.66207890693937e-06,
-5.90163869697242e-05,
-0.000109601357360110,
-0.000158292087960349,
-0.000204973969910915,
-0.000249542810245946,
-0.000291904913907528,
-0.000331977125852081,
-0.000369686834091835,
-0.000404971934638251,
-0.000437780759431793,
-0.000468071968453318,
-0.000495814407316512,
-0.000520986931737601,
-0.000543578200368100,
-0.000563586437558309,
-0.000581019167693471,
-0.000595892922810859,
-0.000608232925264461,
-0.000618072747254343,
-0.000625453949080161,
-0.000630425698012585,
-0.000633044369702842,
-0.000633373134068928,
-0.000631481527607764,
-0.000627445014085407,
-0.000621344535552874,
-0.000613266055623152,
-0.000603300096925844,
-0.000591541274629877,
-0.000578087827892006,
-0.000563041151049750,
-0.000546505326332225,
-0.000528586659811412,
-0.000509393222259934,
-0.000489034396519995,
-0.000467620432921791,
-0.000445262014219172,
-0.000422069831435609,
-0.000398154171935396,
-0.000373624520953468,
-0.000348589177733021,
-0.000323154887333398,
-0.000297426489082019,
-0.000271506582553813,
-0.000245495211870058,
-0.000219489569016230,
-0.000193583716785595,
-0.000167868331862513,
-0.000142430468466854,
-0.000117353342889120,
-9.27161391550566e-05,
-6.85938359691500e-05,
-4.50570549986815e-05,
-2.21719304743364e-05,
-1.87346740234822e-19,
2.14018836163928e-05,
4.19816197769953e-05,
6.16919070940166e-05,
8.04902703878009e-05,
9.83390681632512e-05,
0.000115205481109003,
0.000131061482224025,
0.000145883789232598,
0.000159653800000676,
0.000172357511714387,
0.000183985424624896,
0.000194532431202796,
0.000203997691579672,
0.000212384496184519,

0.000219700116508094,
0.000225955644949308,
0.000231165824714202,
0.000235348870750147,
0.000238526282705587,
0.000240722650909088,
0.000241965456360702,
0.000242284865723817,
0.000241713522296960,
0.000240286333932417,
0.000238040258852403,
0.000235014090293815,
0.000231248240889671,
0.000226784527669252,
0.000221665958529974,
0.000215936521002345,
0.000209640974095084,
0.000202824643971039,
0.000195533224165903,
0.000187812581021308,
0.000179708564961776,
0.000171266828201537,
0.000162532649422522,
0.000153550765919208,
0.000144365213659611,
0.000135019175664793,
0.000125554839062057,
0.000116013261119668,
0.000106434244523717,
9.68562221108855e-05,
8.73161512243955e-05,
7.78494178147670e-05,
6.84897503620943e-05,
5.92691436527458e-05,
5.02177924007311e-05,
4.13640346626618e-05,
3.27343049554072e-05,
2.43530969473100e-05,
1.62429355573125e-05,
8.42435826167659e-06,
9.15905375230231e-07,
-6.26588095666994e-06,
-1.31064493479167e-05,
-1.95932236621213e-05,
-2.57155797470351e-05,
-3.14648147782411e-05,
-3.683410959720 90e-05,
-4.18184844463319e-05,
-4.64147485187423e-05,
-5.06214437536365e-05,
-5.44387833184556e-05,
-5.78685852276504e-05,
-6.09142015539089e-05,
-6.35804436916823e-05,
-6.58735041346801e-05,
-6.78008752287559e-05,
-6.93712653593427e-05,
-7.05945130283952e-05,
-7.14814992697408e-05,
-7.20440588438895e-05,
-7.22948906438355e-05,
-7.22474677322617e-05,
-7.19159474179411e-05,
-7.13150817651340e-05,
-7.04601289144841e-05,
-6.93667655774572e-05,
-6.80510010488354e-05,
-6.65290930632939e-05,
-6.48174658027601e-05,
-6.29326303412047e-05,
-6.08911077928367e-05,
-5.87093554085028e-05,
-5.64036958435200e-05,
-5.39902497983187e-05,
-5.14848722112379e-05,
-4.89030921607121e-05,

-4.62600566120110e-05,
-4.35704781217382e-05,
-4.08485865915686e-05,
-3.81080851412770e-05,
-3.53621101500907e-05,
-3.26231954948451e-05,
-2.99032409934363e-05,
-2.72134850426913e-05,
-2.45644814211058e-05,
-2.19660802089826e-05,
-1.94274127613892e-05,
-1.69568806531068e-05,
-1.45621484994031e-05,
-1.22501405420550e-05,
-1.00270408766353e-05,
-7.89829718465892e-06,
-5.86862782280389e-06,
-3.94203211108448e-06,
-2.12180365257879e-06,
-4.10546509099106e-07,
1.18980594994375e-06,
2.67796971431396e-06,
4.05328650920915e-06,
5.31569807882518e-06,
6.46571857799615e-06,
7.50440527460958e-06,
8.43332776645153e-06,
9.25453591612900e-06,
9.97052670678619e-06,
1.05842102195066e-05,
1.10988749306199e-05,
1.15181525236349e-05,
1.18459824062601e-05,
1.20865761179763e-05,
1.22443818079481e-05,
1.23240489567487e-05,
1.23303935084684e-05,
1.22683635723364e-05,
1.21430058450537e-05,
1.19594328966690e-05,
1.17227914540664e-05,
1.14382318070432e-05,
1.11108784525704e-05,
1.07458020832089e-05,
1.03479930158466e-05,
9.92233614697004e-06,
9.47358751064512e-06,
9.00635250529271e-06,
8.52506584525360e-06,
8.03397328309023e-06,
7.53711513860731e-06,
7.03831166073072e-06,
6.54115023870543e-06,
6.04897446958912e-06,
5.56487507976972e-06,
5.09168268924768e-06,
4.63196239873433e-06,
4.18801017124830e-06,
3.76185097187243e-06,
3.35523862168689e-06,
2.96965731463963e-06,
2.60632473927529e-06,
2.26619674083339e-06,
1.94997345326195e-06,
1.65810682518491e-06,
1.39080945882484e-06,
1.14806467632067e-06,
9.29637723804598e-07,
7.35088020014686e-07,
5.63782353123439e-07,
4.14908926860107e-07,
2.87492154892933e-07,
1.80408100815630e-07,
9.24004599459233e-08,
2.09697848661l4e-08,
-3.19737945852341e-08,

```
                -7.13655272565302e-08,
                -9.76969994672016e-08,
                -1.12634975798380e-07,
                -1.17877156703393e-07,
                -1.15135307104001e-07,
                -1.06118658751313e-07,
                -9.25176799418175e-08,
                -7.59883029998755e-08,
                -5.81366964073535e-08,
                -4.05046645954493e-08,
                -2.45557542337669e-08,
                -1.16621413750432e-08,
                -3.09236906250145e-09,
                0
        };


        /**
         * Nałożenie filtru dolnoprzepustowego na podaną tablicę float
         * @param samples sygnał do przefiltrowania
         * @return przefiltrowany sygnał w postaci tablicy float
         */

        public static float[] filetring(float[] samples) {

                // jeżeli sygnał jest zbyt krótki do filtrowania
                if (samples.length < coeff.length)
                        return null;

                float[] filtered = new float[samples.length + coeff.length];


                //wyliczenie wartości dla sampli od 0 do ostatniego sampla
                for(int i = 0; i < samples.length; i++) {
                        filtered[i] = 0;
                        for(int k = 0; k < coeff.length; k++) {
                                filtered[i] += coeff[k] * samples[i-k];
                                if (i - k == 0) break;
                        }
                }

                //pozostające sample po skończeniu sygnału
                for (int i = samples.length; i < samples.length + coeff.length; i ++) {
                        filtered[i] = 0;
                        for(int k = 0 + (i - samples.length + 1); k < coeff.length; k++) {
                                filtered[i] += coeff[k]*samples[i-k];
                        }

                }

                // zwróć przefiltrowany sygnał
                return filtered;
        }
}
```

Załącznik A3. Kod źródłowy pliku Analysis.java

```
package vsa;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

import org.apache.commons.math3.analysis.interpolation.SplineInterpolator;
import org.apache.commons.math3.analysis.polynomials.PolynomialSplineFunction;

import access.mypackage.offdebug.Debug;

/************************************************
 * Klasa IMFFunction
 *
 * klasa przechowująca wyliczoną funkcję IMF.
```

```
 *
 ************************************************
 */
class IMFFunction {
        /**
         * Funkcja IMF w postaci tablicy float
         */
        float[] samples;
        /**
         * Częstotliwość funkcji IMF
         */
        float freq;

        /**
         * Konstruktor 1. Domyślnie ustawia częstotliwość jako 0.
         * @param size rozmiar funkcji IMF
         */
        IMFFunction(int size) {
                samples = new float[size];
                freq = 0;
        }

        /**
         * Konstruktor 2.
         * @param samples funkcja IMF w postaci tablicy float
         * @param freq wyliczona częstotliwość
         */
        IMFFunction(float [] samples, float freq) {
                this.samples = samples;
                this.freq = freq;
        }
}

/*****************************************************************************
 * Klasa MicrotremorFunction
 *
 * Klasa opisująca funkcje zawierającą mikrodrżenie.
 * Zawiera pole prywatne będące decyzją wykrycia stresu - wartość true;
 * oraz pola zawierające przebieg funkcji i uśrednioną częstotliwość.
 *
 * ***************************************************************************
 */
class MicrotremorFunction extends IMFFunction {

        /**
         * Zmienna określająca czy stres został wykryty
         */
        private boolean stressDetected = false;

        /**
         * Konstruktor.
         * @param samples tablica przedstawiająca funkcję znajdującą się najbliżej mikrodrżenia
         * @param freq częstotliwość funkcji
         * @param stressDetected stwierdzenie występowania stresu
         */
        MicrotremorFunction(float[] samples, float freq, boolean stressDetected) {
                super(samples, freq);
                this.stressDetected = stressDetected;
        }

        /**
         * Zwraca informacje czy w danej funkcji stwierdzono występowanie stresu.
         * @return zmienna typu boolean określająca występowanie stresu
         */
        public boolean getDecision(){
                return stressDetected;
        }
}

/*****************************************************************************
 * Klasa AnalysisInterruptedException
 *
 * Wyjątek stworzony do obsługi przerwania analizy.
 * Dziedziczy po klasie Exception.
 *
 * ***************************************************************************
```

```
 */
@SuppressWarnings("serial")
class AnalysisInterruptedException extends Exception {

	public AnalysisInterruptedException() {
		Debug.debug("AnalysisInterruptedException");
	}
}

/**************************************************************************
 * Klasa Zeros
 *
 * Klasa zawierająca liczbę oraz położenie miejsc zerowych funkcji
 *
 **************************************************************************
 */
class Zeros {
	/**
	 * Liczba miejsc zerowych
	 */
	int nrOfZeros;
	/**
	 * Położenie miejsc zerowych
	 */
	ArrayList<Integer> zeroPoints;
	/**
	 * Konstruktor
	 */
	public Zeros(){
		nrOfZeros = 0;
		zeroPoints = new ArrayList<Integer>();
	}
}

/**************************************************************************
 * Klasa Analysis
 *
 * Klasa wykonująca algorytm EMD oraz szereg operacji z nim związanych.
 *
 **************************************************************************
 */
public class Analysis {

	/**------------------------------------------------------------------
	 *
	 * Typy wyliczeniowe
	 *
	 * ------------------------------------------------------------------
	 */

	/**
	 * Typ wyliczeniowy określający stan w jakim jest proces analizy
	 * Umożliwia przerwanie wykonywania
	 */
	enum AnalysisState {
		INACTIVITY, ANALYSING, INTERRUPTING
	}

	/**
	 * Typ wyliczeniowy określający tendencję badanej funkcji
	 */
	enum Tendency {
		INCREASING, DECREASING, NO_TENDENCY
	}

	/**------------------------------------------------------------------
	 *
	 * Pola prywatne
	 *
	 * ------------------------------------------------------------------
	 */

	// częstotliwość próbkowania
	private static final float sampleRate =
(float)AudioFileOperations.getAudioFormat().getSampleRate();
```

```
        // przedział tolerancji eliminujący szum kwantyzacji ->  wartość jednego bita zrzutowana na
liczbę typu float
        private static final float delta = 1.0F/(float)(Math.pow(2.0, 15) );

        // zmienna określająca stan w jakim znajduje sie proces analizy
        private volatile AnalysisState analysisState = AnalysisState.INACTIVITY;

        // obiekt służący do synchronizacji zmiany stanu procesu analizy
        private Object analysisLock = new Object();

        /**-------------------------------------------------------------------------
         *
         * Metody publiczne
         *
         * -------------------------------------------------------------------------
         */

        /**
         * Rozpoczęcie analizy wyszukiwania mikrodrżenia.
         * @param samples tablica float z audio, na którym ma być przeprowadzona analiza
         * @return funkcja najbliższa mikrodrżeniu
         * @throws InterruptedException przerwanie analizy
         */
        public MicrotremorFunction startAnalysis(float[] samples) throws AnalysisInterruptedException{
                //Mapa zawierająca nr porządkowy, oraz klasę IMFFunction
                HashMap<Integer, IMFFunction> imfMap = new HashMap<Integer, IMFFunction>();

                //Ustaw stan procesu analizy na Analysing
                setAnalysisState(AnalysisState.ANALYSING);

                // Nałóż filtr dolnoprzepustowy. Jeżeli długość próbek była zbyt krótka
                // Funkcja zwraca null i jest przerywany proces analizy
                if((samples = LowPassFilter.filetring(samples)) == null)
                        setAnalysisState(AnalysisState.INTERRUPTING);;
                if(getAnalysisState() == AnalysisState.INTERRUPTING)
                        throw new AnalysisInterruptedException();

                // Wykonaj algorytm EMD
                emd(samples, imfMap);
                if(getAnalysisState() == AnalysisState.INTERRUPTING)
                        throw new AnalysisInterruptedException();

                // Zwróć funkcję najbliższą mikrodrżeniu
                return findClosestFrequencyToMicrotremor(imfMap);
        }

        /**-------------------------------------------------------------------------
         *
         * Gettery i settery
         *
         * -------------------------------------------------------------------------
         */

        /**
         * Zmiana stanu w jakim znajduje się proces analizy
         * @param state stan jaki ma przyjąć zmienna
         */
        public void setAnalysisState(AnalysisState state) {
                synchronized(analysisLock) {
                        analysisState = state;
                }
        }

        /**
         * Zwróć stan procesu analizy
         * @return stan procesu analizy
         */
        public AnalysisState getAnalysisState() {
                return analysisState;
        }


        /**-------------------------------------------------------------------------
         *
         * Metody prywatne
         *
```

```
         * ------------------------------------------------------------------------
         */

        /**
         * Znalezienie kolejnych funkcji IMF
         * @param residuum sygnał wejściowy
         * @param imfMap mapa, w której może być przechowany numer oraz postać funkcji IMF
         * @throws AnalysisInterruptedException wyjątek przerywający działanie metody
         */
        private void emd(float[] residuum, HashMap<Integer, IMFFunction> imfMap) throws
AnalysisInterruptedException{

                // zmienne przeznaczone do przechowywania górnej i dolnej obwiedni
                float[] minEnvelope = new float[residuum.length];
                float[] maxEnvelope= new float[residuum.length];

                // sprawdzenie czy jest odpowiednia liczba sampli
                if(residuum.length < 6)
                        throw new AnalysisInterruptedException();

                // zmienna określająca odchylenie standardowe
                float squaredDifference;

                // listy zawierające położenie minimów i maksimów
                ArrayList<Integer> max = new ArrayList<Integer>();
                ArrayList<Integer> min = new ArrayList<Integer>();

                // zmienna przechowujące kopie sygnału
                // do iteracji wewnątrz pętli while (kolejne komponenty sygnału)
                float[] component = new float[residuum.length];

                // zmienna na przechowywanie średniej z obwiedni
                float[] meanEnvelope = new float[component.length];

                // zmienna do wyliczania odchylenia standardowego
                // potrzebny jest sygnał przed i po odjęciu średniej z obwiedni
                float[] previousComponent = new float[component.length];

                // zewnętrzna pętla wykonująca się aż do przerwania
                outer : while(true) {
                        // skopiuj sygnał wejściowy do komponentu
                        component = residuum.clone();

                        //pętla wewnętrzna
                        do {

                                //zresetuj listy zawierające minima i maksima
                                max.clear();
                                min.clear();

                                //znajdź ekstrema funkcji component
                                extremum(max, min, component);

                                if(getAnalysisState() == AnalysisState.INTERRUPTING)
                                        throw new AnalysisInterruptedException();

                                // warunki przerywające pętle zewnętrzną
                                // punkty na krawędziach przedziałów są przyjmowane jako ekstrema
(stany nieustalone funkcji)
                                if(max.size() < 3) break outer;
                                if(min.size() < 3) break outer;

                                // interpolacja maksimów i minimów za pomocą funkcji sklejanych 3
rzędu
                                maxEnvelope = interpolation(component, max);
                                minEnvelope = interpolation(component, min);

                                if(getAnalysisState() == AnalysisState.INTERRUPTING)
                                        throw new AnalysisInterruptedException();

                                // wyliczenie średniej z obwiedni
                                for(int i = 0; i < component.length; i++)
                                        meanEnvelope[i] = (maxEnvelope[i]+minEnvelope[i])/2;

                                // zapamiętanie obecnego komponentu
                                previousComponent = component.clone();
```

```
                                // odjecie średniej z obwiedni od komponentu
                                for(int i = 0; i < component.length; i++)
                                        component[i] -= meanEnvelope[i];

                                // wyzerowanie odchylenia standardowego
                                squaredDifference = 0;

                                //obliczenie odchylenia standardowego
                                for(int i =0; i < component.length; i++)
                                        squaredDifference += (component[i]-
previousComponent[i])*(component[i]-
previousComponent[i])/(previousComponent[i]*previousComponent[i]+1e-8);

                                if(getAnalysisState() == AnalysisState.INTERRUPTING)
                                        throw new AnalysisInterruptedException();

                                // warunek kończący pętlę wewnętrzną
                        } while(squaredDifference > 5);

                        //sprawdzenie częstotliwości otrzymanej funkcji
                        float freq = avgerageFrequency(component);

                        // jeżeli jej częstotliwość jest już na tyle niska, że nie ma
                        // to wpływu na wykrywanie mikrodrżenia można przerwać obliczenia
                        if (freq < 4F)
                                break;                  // przerywa pętlę zewnętrzną

                        // jeżeli udało się otrzymać częstotliwość to
                        // podana funkcja component będzie zapisana jako funkcja IMF
                        IMFFunction imfFunction = new IMFFunction(component.clone(), freq);

                        if(getAnalysisState() == AnalysisState.INTERRUPTING)
                                throw new AnalysisInterruptedException();

                        // dodaj do mapy znalezioną funkcję IMF
                        imfMap.put(imfMap.size(), imfFunction);

                        //odjecie od sygnału początkowego funkcji IMF
                        for(int i =0; i <residuum.length; i++) {
                                residuum[i] -= component[i];
                        }
                } // koniec pętli zewnętrznej
        } // koniec metody emd

        /**
         * Metoda interpolująca sygnał funkcjami sklejanymi trzeciego rzędu, na podstawie podanych
punktów.
         * Lista określa położenie w osi czasu punktów.
         * Ich wartości są pobierane z tablicy samples.
         * @param samples funkcja, z której mają być wczytane wartości na osi y
         * @param list lista położenia punktów na osi x
         * @return wynik interpolacji
         */
        private float[] interpolation(float[] samples, ArrayList<Integer> list) {

                // interpolowana funkcja
                float[] interpolatedFunction = new float[samples.length];

                //Iterator do poruszania się po liście
                Iterator<Integer> iterator = list.iterator();

                //punkty, na podstawie których ma być dokonana interpolacja
                double[] x = new double[list.size()];
                double[] y = new double[list.size()];

                int j = 0;
                int index = 0;
                // jeśli jest następny punkt
                while(iterator.hasNext()) {
                        // zapamiętaj indeks tego punktu
                        index = iterator.next();
                        // indeks jest położeniem próbki na osi x
                        x [j] = (double)index;
                        // wartość próbki jest przechowywana w tablicy samples
                        y [j] = (double)samples[index];
```

```
                        j++;
                }

                // stworzenie obiektu wyliczającego funkcje służące do interpolacji
                SplineInterpolator interpolator = new SplineInterpolator();
                // zbiór funkcji na podstawie, których jest określana wartość dowolnego punktu na osi x
                PolynomialSplineFunction poly;
                poly = interpolator.interpolate(x, y);

                // obliczenie wartości dla każdej próbki w sygnale
                for(int i = 0; i < samples.length; i ++)
                        interpolatedFunction[i] = (float) poly.value((double)i);

                return interpolatedFunction;
        } // koniec metody interpolation

        /**
         * Metoda obliczającą średnią częstotliwość funkcji na podstawie miejsc zerowych
         * @param samples sygnał, z którego ma być obliczona średnia częstotliwość
         * @return średnia częstotliwość
         * @throws AnalysisInterruptedException wyjątek przerywający działanie metody
         */
        private float avgerageFrequency(float[] samples) throws AnalysisInterruptedException{

                float freq = 0F;

                // znalezienie miejsc zerowych w sygnale
                Zeros zeros = findingZeros(samples);

                // gdy nieparzysta liczba zer
                if((zeros.nrOfZeros % 2)!= 0) {
                        if (zeros.nrOfZeros >= 3)
                                // jeżeli więcej niż lub równe trzy to odejmij jedno od liczby zer
                                // w ten sposób dostaje się liczbę podwójnych przejść przez zero
sygnału
                                // (czyli podwojoną liczbę okresów),
                                // następnie ta liczba jest dzielona przez ostatnie zero
                                // odjęte od pierwszego i odpowiednio
                                // wymnaża się przez częstotliwość próbkowania.
                                freq = (zeros.nrOfZeros -
1)/(float)(zeros.zeroPoints.get(zeros.nrOfZeros - 1) - (float)zeros.zeroPoints.get(0))/2.0F*sampleRate;
                        else
                                // jeżeli jest mniej niż 3 zera to nie da się wyliczyć częstotliwości
                                // (jedno miejsce zerowe)
                                freq = 0;
                }
                // liczba zer jest parzysta
                else {
                        if(zeros.nrOfZeros >= 4)
                                //jeżeli liczba zer jest większa niż lub równa cztery
                                // to pomiń ostatnie zero w obliczaniu częstotliwości
                                freq = (zeros.nrOfZeros -
2)/(float)(zeros.zeroPoints.get(zeros.nrOfZeros - 2) - zeros.zeroPoints.get(0))/2.0F*sampleRate;
                        else if (zeros.nrOfZeros == 2)
                                // jeżeli liczba zer jest równa dwa
                                // to powiel różnicę odległości drugiego zera i pierwszego zera,
                                // żeby mieć pełen okres funkcji
                                freq =
(zeros.nrOfZeros)/(float)(((zeros.zeroPoints.get(zeros.nrOfZeros - 1) -
(zeros.zeroPoints.get(0)))*2))/2.0F*sampleRate;
                        else
                                freq = 0;

                        if(getAnalysisState() == AnalysisState.INTERRUPTING)
                                throw new AnalysisInterruptedException();
                }
                Debug.debug(Float.toString(freq));

                return freq;
        } // koniec metody averageFrequency

        /**
         * Znalezienie ekstremów w sygnale.
         * @param max lista liczb całkowitych określająca położenie maksimów
         * @param min lista liczb całkowitych określająca położenie minimów
         * @param samples sygnał, w którym mają być znalezione ekstrema
```

```
	 */
	private void extremum(ArrayList<Integer> max, ArrayList<Integer> min, float[] samples) {

		// tendencja funkcji
		Tendency tendency = Tendency.NO_TENDENCY;

		//pierwszy punkt traktowany jako stan nieustalony, jednocześnie jako maksimum i jako
minimum
		max.add(0);
		min.add(0);

		//Sprawdzenie tendencji funkcji na początku sygnału
		int i;
		for(i = 0; i < samples.length - 1 && tendency == Tendency.NO_TENDENCY; i++) {
			if (samples[i] > delta/2)
				tendency = Tendency.INCREASING;
			else if (samples[i] < - delta/2)
				tendency = Tendency.DECREASING;
		}

		//jeżeli funkcja jest płaska to nie ma żadnej tendencji
		//należy dodać ostatnie punkty jako ekstrema i wyjść z funkcji
		if (tendency == Tendency.NO_TENDENCY) {
			min.add(samples.length -1);
			max.add(samples.length -1);
			return;
		}

		//tymczasowe maksimum i minimum, ich wartość będzie się zmieniać wraz z badaniem
funkcji
		int maxTemp = 0;
		int minTemp = 0;

		// Pierwsze przejście musi być w innej pętli, bo może sie zdarzyć tak,
		// że dwa razy zapisze się ta sama wartość na osi x jako minimum albo maksimum.
		// Jeżeli przykładowo pierwsza wartość w sygnale jest wystarczająco duża,
		// żeby funkcja przyjęła tendencję rosnącą,
		// a następna wartość jest wystarczająco mała, żeby algorytm stwierdził ekstremum,
		// to nie może zapisać tej większej wartości jako maksimum, ponieważ
		// raz już zostało zapisane. (powstał by problem w funkcji interpolate)
		// Ta sytuacja występuje tylko do momentu, w którym dla tendencji rosnącej zmieniła się
		// wartość tymczasowego maksimum, albo zmieniła sie tendencja funkcji,
		// a dla tendencji malejącej zmieniła się wartość minimum,
		// albo zmieniła się tendencja funkcji.

		//tendencja rosnąca
		for (; i < samples.length - 1; i++) {
			if(tendency == Tendency.INCREASING) {
				// jeżeli następna próbka jest większa to weź ją jako maksimum
				if(samples[maxTemp] < samples[i]) {
					maxTemp = i;
					break;
				}
				//obecne maksimum jest większe o deltę
				else if (samples[maxTemp] > (samples[i] + delta)) {
					minTemp = i;
					tendency = Tendency.DECREASING;
					break;
				}
				//nie jest większy o deltę - szukaj dalej
				else {
					//maxTemp zostawiam bez zmian
					minTemp = i;
				}
			//tendencja malejąca
			} else {
				//jeżeli następna próbka jest mniejsza to weź ją jako minimum
				if(samples[minTemp] > samples[i]) {
					minTemp = i;
					break;
				}
				//jeżeli obecne minimum jest mniejsze o deltę
				else if (samples[minTemp] < (samples[i] - delta)) {
					maxTemp = i;
					tendency = Tendency.INCREASING;
					break;
```

```
                        }
                        else {
                                maxTemp = i;
                        }
                }
        }

        // dalsza część odbywa się już normalnie
        for(; i < samples.length - 1; i++) {
                //tendencja rosnąca
                if(tendency == Tendency.INCREASING) {
                        // jeżeli następna próbka jest większa to weź ją jako maksimum
                        if(samples[maxTemp] < samples[i])
                                maxTemp = i;
                        //obecne maksimum jest większe o deltę
                        else if (samples[maxTemp] > (samples[i] + delta)) {
                                max.add(maxTemp);
                                minTemp = i;
                                tendency = Tendency.DECREASING;
                        }
                        //nie jest większe o deltę - szukaj dalej
                        else {
                                minTemp = i;
                        }
                //tendencja malejąca
                } else {
                        //jeżeli następna próbka jest mniejsza od to weź ją jako minimum
                        if(samples[minTemp] > samples[i])
                                minTemp = i;
                        //jeżeli obecne minimum jest mniejsze o deltę
                        else if (samples[minTemp] < (samples[i] - delta)) {
                                min.add(minTemp);
                                maxTemp = i;
                                tendency = Tendency.INCREASING;
                        }
                        else {
                                maxTemp = i;
                        }
                }


        }

        //Dodaj ostatnią próbkę
        min.add(samples.length - 1);
        max.add(samples.length - 1);
} // koniec metody ekstremum

/**
 * Wyszukiwanie miejsc zerowych w sygnale
 * @param samples sygnał, w którym mają być znalezione miejsca zerowe
 * @return Zeros - obiekt przechowujący położenie oraz liczbę wartości miejsc zerowych
 */
private Zeros findingZeros(float[] samples) {
        Zeros zeros = new Zeros();

        zeros.nrOfZeros = 0;
        zeros.zeroPoints.clear();

        Tendency tendency = Tendency.NO_TENDENCY;

        int i;

        //Jeżeli funkcja nie ma jeszcze określonej tendencji sprawdzaj czy wartość kolejnej
próbki
        //nie przekroczyła zadanego przedziału tolerancji od wartości zerowej
        for(i = 0; i <(samples.length - 1) && (tendency == Tendency.NO_TENDENCY); i++) {
                if (samples[i] > delta/2)
                        tendency = Tendency.INCREASING;
                if(samples[i] < -delta/2)
                        tendency = Tendency.DECREASING;
        }

        //Jeżeli funkcja ma już określoną tendencję kontynuuj wyszukiwanie, ale za miejsce
zerowe weź
        //miejsce w którym próbka przekroczy oś o wyznaczoną tolerancję
```

```
                for(; i < samples.length-1; i++) {
                        if (tendency == Tendency.INCREASING) {
                                if (samples[i] < -delta) {
                                        zeros.nrOfZeros++;
                                        zeros.zeroPoints.add(i);
                                        tendency = Tendency.DECREASING;
                                }
                        }
                        else if(tendency == Tendency.DECREASING){
                                if (samples[i] > delta) {
                                        zeros.nrOfZeros++;
                                        zeros.zeroPoints.add(i);
                                        tendency = Tendency.INCREASING;
                                }
                        }
                }
                return zeros;
        } // koniec metody findingZeros

        /**
         * Określenie częstotliwości najbliżej mikrodrżenia spośród wszystkich otrzymanych funkcji IMF.
         * @param imfMap mapa funkcji IMF
         * @return klasa MicrotremorFunction przechowująca funkcję IMF najbliższą mikrodrżeniu
         */
        private MicrotremorFunction findClosestFrequencyToMicrotremor(HashMap<Integer, IMFFunction>
imfMap) {

                // iterator mapy
                Iterator<Map.Entry<Integer, IMFFunction>> iterator = imfMap.entrySet().iterator();
                Map.Entry<Integer, IMFFunction> mapEntry;

                // częstotliwość mikrodrżenia
                float microtremorFrequency;
                int id = 0;

                // przyjęcie pierwszej funkcji jako mikrodrżenia
                if(iterator.hasNext()) {
                        mapEntry = iterator.next();
                        microtremorFrequency = mapEntry.getValue().freq;
                        id = 0;
                }
                else {
                        Debug.debug("nie ma zadnej czestotliwosci do analizowania w funkcji
findClosestFrequencyToMicrottremor()");
                        return null;
                }

                while(iterator.hasNext()) {
                        mapEntry = iterator.next();

                        //znalezienie ID funkcji IMF najbliższej częstotliwości mikrodrżenia
                        if(Math.abs(10 - mapEntry.getValue().freq) < Math.abs(10 -
microtremorFrequency)) {
                                microtremorFrequency = mapEntry.getValue().freq;
                                id = mapEntry.getKey();
                        }
                }
                // zwróć funkcję odpowiadającą mikrodrżeniu, jej częstotliwość  oraz podjętą decyzję.
                return new MicrotremorFunction((imfMap.get((Integer)id).samples), microtremorFrequency,
decision(microtremorFrequency));
        } // koniec metody findClosestFrequencyToMicrotremor

        /**
         * Podjęcie decyzji odnośnie występowania stresu
         * @param frequency częstotliwość funkcji IMF
         * @return decyzja
         */
        private boolean decision(float frequency) {
                //wykryto stres
                if (frequency < 8 || frequency > 12)
                        return true;
                //brak stresu
                else
                        return false;
        }
} // koniec klasy Analysis
```

Załącznik B. Płyta CD z programem i jego kodem źródłowym