Raport 2

Generowanie dźwięków w Scilabie i Arduino IDE

Agnieszka Staszkiewicz 268791, Wiktoria Tęcza 267637

23.06.2024

Spis treści

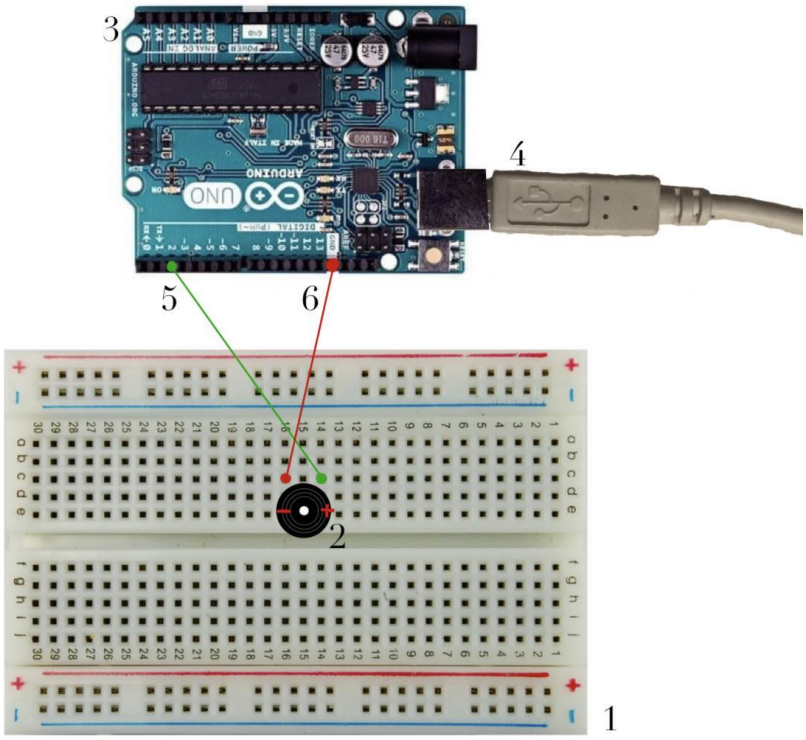
1 Wstęp

1.1 Cel raportu

Celem niniejszego raportu jest zaprezentowanie procesu generowania sygnałów dźwiękowych przy użyciu Arduino i Scilab oraz przeprowadzenie ich szczegółowej analizy. Projekt obejmuje budowę prostego obwodu z Arduino i buzzerem oraz implementację kodów w Scilab i Arduino IDE do tworzenia i odtwarzania melodii. W dalszej części, raport skupia się na analizie wygenerowanych sygnałów dźwiękowych, porównaniu spektrogramów i transformacji Fouriera, a także na wpływie modulacji amplitudy na te sygnały.

2 Konstrukcja obwodu z Arduino i głośnikiem

W celu wygenerowania sygnału dźwiękowego w Arduino IDE musimy najpierw skonstruować obwód za pomocą Arduino i głośnika. Poniżej znajduję się schemat obwodu i opis rzeczy wykorzystanych do jego budowy.



Rysunek 1: Schemat obwodu z Arduino i głośnikiem.

Oznaczenia:

1. Płytka stykowa (breadboard) - znajdują się w niej rzędy i kolumny otworów, które są wewnętrznie połączone, umożliwiając łatwe łączenie elementów elektronicznych.

2. Głośnik (buzzer) - przekształca sygnały elektryczne w dźwięki.

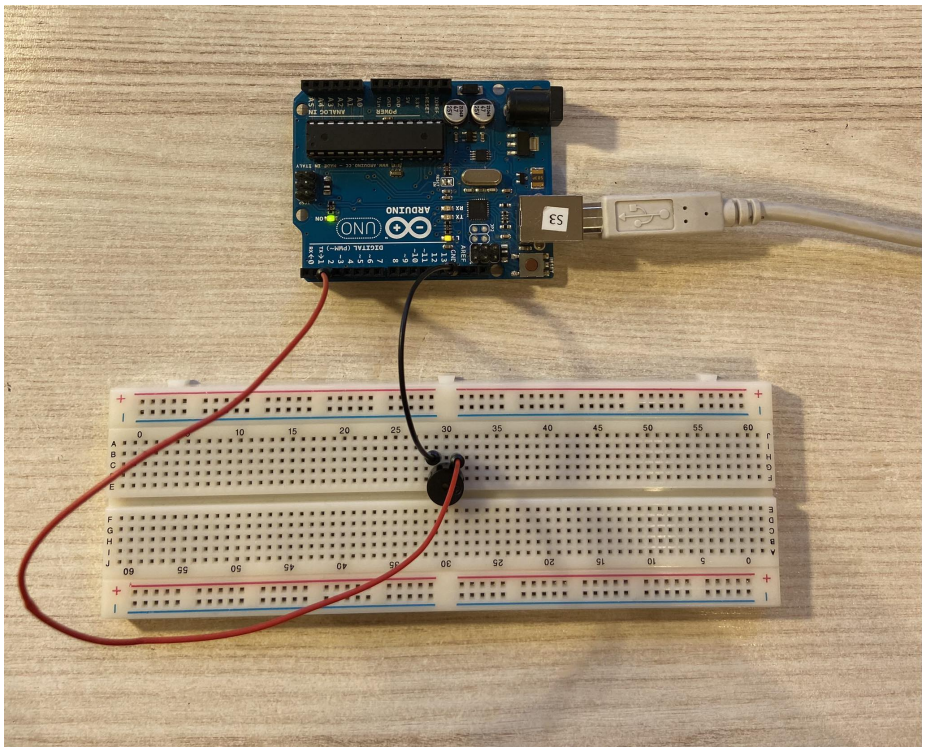
3. Arduino UNO - mikrokontroler, który steruje działaniem obwodu.

4. Kabel USB - służy do podłączenia Arduino z komputerem.

5. Przewód (zielony) - łączy pin 2 z Arduino UNO z dodatnim (+) wejściem głośnika.

6. Przewód (czerwony) - łączy pin GND z Arduino UNO z negatywnym (−) wejściem głośnika.

Korzystając z powyższego schematu możemy zbudować obwód Arduino z głośnikiem. Poniżej przedstawiono zdjęcie obwodu Arduino z głośnikiem skonstuowanego w domu.



Rysunek 2: Zdjęcie obwodu z Arduino i głośnikiem.

3 Sygnał dźwiękowy (melodia) w Scilabie

3.1 Funkcja w Scilabie

Pierwsze dwa dźwięki wygenerujemy za pomocą Scilaba. Pierwszy z nich będzie dźwiękem 'czystym' a drugi zanieczyszczony szumem. Poniżej znajduję się początek skryptu do generowania obu dźwięków, który zawiera dwa wektory. Pierwszym z nich jest wektor definicji częstotliwości nut używanych w melodii a drugi wektor zawiera definicje melodi oraz odpowiednie czasy ich trwania. To właśnie drugi wektor odpowiada za wybraną przez nas melodie, którą chcemy wygenerować.

Fragment kodu z pliku ' scilab.bez.szumu.sci'

```
// Definicje czestotliwosci nut uzywanych w melodii (na podstawie
    kodu Arduino)
NOTES = [
  31,   // NOTE_B0
  33,   // NOTE_C1
  35,   // NOTE_CS1
  37,   // NOTE_D1
  39,   // NOTE_DS1
  .
  .
  .
  3951, // NOTE_B7
  4186, // NOTE_C8
  4435, // NOTE_CS8
  4699, // NOTE_D8
  4978, // NOTE_DS8
  0     // REST
]

// Definicja melodii i jej czasow trwania (na podstawie kodu
    Arduino)
melody = [
  60, 8; // NOTE_FS5,8
  60, 8; // NOTE_FS5,8
  58, 8; // NOTE_D5,8
  47, 8; // NOTE_B4,8
  .
  .
  .
  60, 8; // NOTE_FS5,8
  60, 8; // NOTE_FS5,8
  58, 8; // NOTE_D5,8
  47, 8  // NOTE_B4,8
]

// Wyodrebnienie indeksow nut oraz ich czas trwania do osobnych
    wektorow
```

```
noteIndices = melody(:, 1) // Pierwsza kolumna: indeksy nut
durations = melody(:, 2)   // Druga kolumna: dlugosci trwania nut

// Ustawienie tempa
tempo = 100

// Obliczenie calej dlugosci
wholenote = (60000 * 4) / tempo

// Inicjalizacja czestotliwosci probkowania
fs = 44100 // Czestotliwosc probkowania
```

3.2 Generowanie dźwięku czystego

Aby wygenerować dźwięk 'czysty' skorzystamy z funkcji 'generateNoteFrequency' która generowuje sygnał dźwiękowy dla danej nuty przez określony czas. Funkcja ta przyjmuje dwa argumenty. Pierwszym z nich jest częstotliwość nuty a drugim czas trwania nuty. W poniższym fragmencie kodu generujemy melodie i zapisujemy ją do pliku 'melody.clear.wav'.

Fragment kodu z pliku ' scilab.bez.szumu.sci'

```

// Funkcja generujaca sygnal nuty
function signal = generateNoteFrequency(note, duration)
    if note <> 0 then
        t = (0:1/fs:duration/1000)' // Wektor czasu
        signal = sin(2 * %pi * note * t) // Generowanie sygnalu
    nuty
    else
        signal = zeros(fs * duration / 1000, 1)  // Generowanie
    ciszy dla pauz
    end
endfunction


// Inicjalizacja pustej tablic na cala melodie
melodySignal = []


// Generowanie melodii
for i = 1:length(noteIndices)
    noteIndex = noteIndices(i)
    note = NOTES(noteIndex + 1) // Dostosowanie indeksu, poniewaz w
     Scilabie indeksowanie jest od 1
    divider = durations(i)
    if divider > 0 then
        noteDuration = wholenote / divider
    else
        noteDuration = (wholenote / abs(divider)) * 1.5
    end

```

```
    signal = generateNoteFrequency(note, noteDuration)
    melodySignal = [melodySignal; signal] // Dodanie sygnalu do
    melodii
end

// Normalizacja sygnalu melodii do zakresu [-1, 1]
melodySignal = melodySignal / max(abs(melodySignal))


// Zapis melodii do pliku WAV
filename_clear = 'C:\Users\gusia\Desktop\semestr4\analiza sygnalow\
    raport 2\melody.clear.wav';
wavwrite(melodySignal, fs, filename_clear)

disp("Melodia zapisana do " + filename_clear)

// Odtworzenie melodii
sound(melodySignal, fs)
```

3.3 Generowanie dźwięku zanieczyszczonym szumem

Aby wygenerować dźwięk zanieczyszony szumem z rozkładu normalnego $\mathcal{N}(0,1)$ skorzystamy z funkcji 'generateNoisyNoteFrequency' która generuje sygnał dźwiękowy z szumem dla danej nuty przez określony czas. Funkcja ta przyjmuje dwa argumenty. Pierwszym z nich jest częstotliwość nuty a drugim czas trwania nuty. W poniższym fragmencie kodu generujemy melodie z szumem i zapisujemy ją do pliku 'melody.with.noise.wav'.

Fragment kodu z pliku ' scilab.szum.sci'

```

// Funkcja generujaca sygnal nuty z szumem
function signal = generateNoisyNoteFrequency(note, duration)
    if note <> 0 then
        t = (0:1/fs:duration/1000)' // Wektor czasu
        signal = sin(2 * %pi * note * t) // Generowanie sygnalu dla
     nuty
        noise = grand(length(t), 1, "nor", 0, 1) // Generowanie
    bialego szumu z rozkladu N(0,1)
        signal = signal + 0.2 * noise // Dodanie sygnalu z szumem
    else
        signal = zeros(fs * duration / 1000, 1) // Generowanie
    ciszy dla pauz
    end
endfunction

// Inicjalizacja pustej tablic na cala melodie
melodySignalNoisy = []

// Generowanie melodii
for i = 1:length(noteIndices)
```

```
    noteIndex = noteIndices(i)
    note = NOTES(noteIndex + 1) // Dostosowanie indeksu, poniewaz w Scilabie indeksowanie jest od 1
    divider = durations(i)
    if divider > 0 then
        noteDuration = wholenote / divider
    else
        noteDuration = (wholenote / abs(divider)) * 1.5
    end

    signalNoisy = generateNoisyNoteFrequency(note, noteDuration)
    melodySignalNoisy = [melodySignalNoisy; signalNoisy] // Dodanie sygnalu do melodii
end

// Normalizacja sygnalu melodii do zakresu [-1, 1]
melodySignalNoisy = melodySignalNoisy / max(abs(melodySignalNoisy))


// Zapis melodii do pliku WAV
filename_noisy = 'C:\Users\gusia\Desktop\semestr4\analiza sygnalow\raport 2\melody.with.noise.wav'

wavwrite(melodySignalNoisy, fs, filename_noisy)

disp("Melody with noise saved to " + filename_noisy)


// Odtworzenie melodii

sound(melodySignalNoisy, fs)
```

4 Sygnał dźwiękowy (melodia) w Arduino IDE

Aby wygenerować dźwięk za pomocą Arduino IDE musimy zbudować odwód Arduino z głośnikiem oraz napisać odpowiedni skrypt w Arduino IDE, który będzie zawierał informacje jaką melodie chcemy wygenerować. Na początku raportu pokazaliśmy schemat konstrukcji obwodu z Arduino i głośnikiem. Teraz pokażemy kod w Arduino IDE dzięki któremu będziemy mogli wygenerować melodie.

4.1 Funkcja w Arduino IDE

Poniższy skrypt pozwala wygenerować melodie, która zostaje odtworzona przez głośnik z obwodu z Arduino. Melodia została nagrana przez telefon i zapisana w pliku ' melody.arduino.wav'.

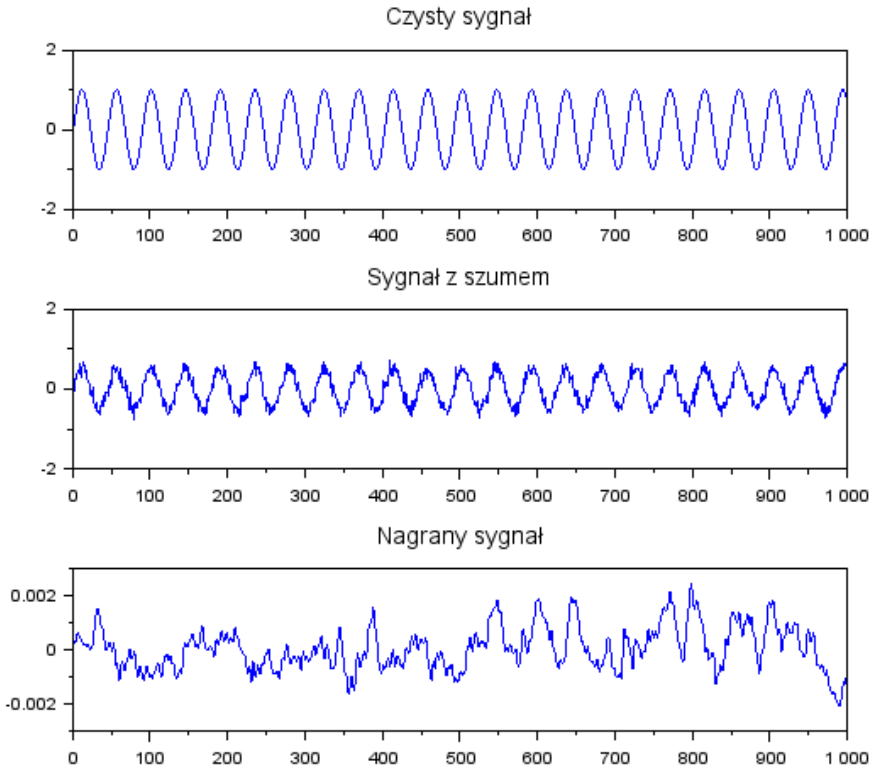
Fragment kodu z pliku ' melody.arduino.ino'

```
1
2  // Take on me, wykonawca : A-ha
3
4  #define NOTE_B0  31
5  #define NOTE_C1  33
6  #define NOTE_CS1 35
7  #define NOTE_D1  37
8  #define NOTE_DS1 39
9  .
10 .
11 .
12 #define NOTE_C8  4186
13 #define NOTE_CS8 4435
14 #define NOTE_D8  4699
15 #define NOTE_DS8 4978
16 #define REST      0
17
18
19 // Tempo utworu
20 int tempo = 100
21
22 // Podpiety pin
23 int buzzer = 2
24
25 //Wektor zawiera czestotliwosc nuty i czas trwania
26 int melody[] = {
27
28   // 8 - oznacza osemke w nutach
29
30   NOTE_FS5,8, NOTE_FS5,8,NOTE_D5,8, NOTE_B4,8, REST,8, NOTE_B4,8,
       REST,8, NOTE_E5,8,
31   REST,8, NOTE_E5,8, REST,8, NOTE_E5,8, NOTE_GS5,8, NOTE_GS5,8,
       NOTE_A5,8, NOTE_B5,8,
```

```
  NOTE_A5,8, NOTE_A5,8, NOTE_A5,8, NOTE_E5,8, REST,8, NOTE_D5,8,
    REST,8, NOTE_FS5,8,
  REST,8, NOTE_FS5,8, REST,8, NOTE_FS5,8, NOTE_E5,8, NOTE_E5,8,
    NOTE_FS5,8, NOTE_E5,8,
  NOTE_FS5,8, NOTE_FS5,8,NOTE_D5,8, NOTE_B4,8


};

//Rozmiar w bajtach
int notes = sizeof(melody) / sizeof(melody[0]) / 2

// Dlugosc w ms
int wholenote = (60000 * 4) / tempo

int divider = 0, noteDuration = 0

void setup() {

  // iteruje po nutach melodii
  for (int thisNote = 0;thisNote < notes * 2; thisNote = thisNote +
     2) {

    // Oblicza dlugosc kazdej nuty
    divider = melody[thisNote + 1]
    if (divider > 0) {
      noteDuration = (wholenote) / divider
    } else if (divider < 0) {
      noteDuration = (wholenote) / abs(divider)
      noteDuration *= 1.5
    }

    tone(buzzer, melody[thisNote], noteDuration)

    delay(noteDuration)

    noTone(buzzer)
  }
}

void loop() {
  // nie trzeba powtarzac melodii
}
```

5 Porównanie spektogramów dla trzech sygnałów

W ramach analizy mamy trzy rodzaje sygnałów dźwiękowych: czysty dźwięk wygenerowany w Scilabie, ten sam sygnał z szumem oraz dźwięk wygenerowany przez Arduino i zarejestrowany przez mikrofon. Każdy z tych sygnałów przedstawiony jest za pomocą wykresu na poniższym rysunku.



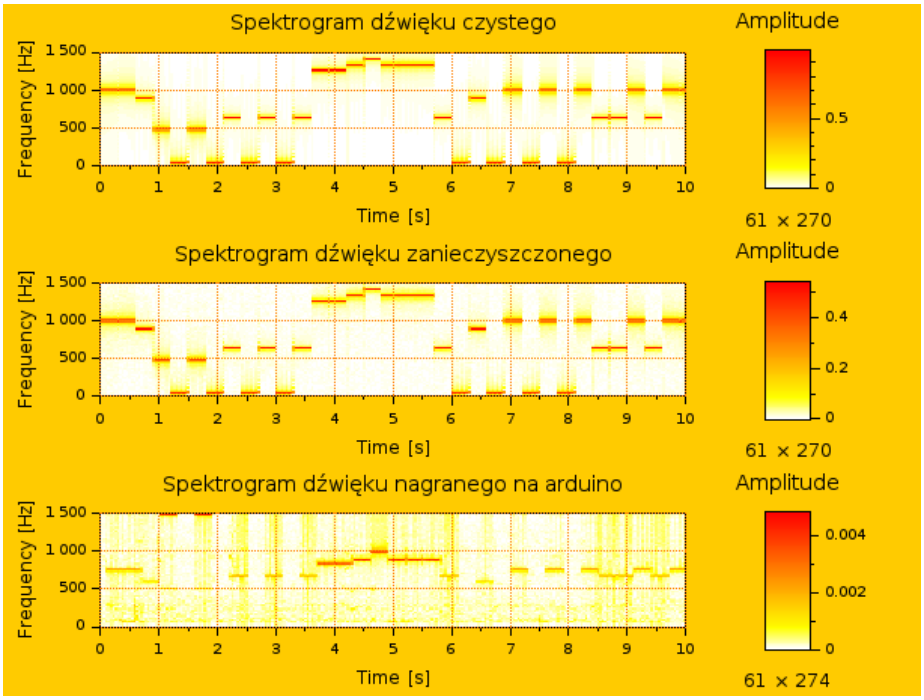
Rysunek 3: .

Fragment kodu z pliku ' wykresy.sygnalow.sci'

```
    // Sciezki do plikow z melodia bez szumu i z szumem
filename_clear = 'C:\Users\gusia\Desktop\semestr4\analiza sygnalow\
    raport 2\melody.clear.wav'
filename_noisy = 'C:\Users\gusia\Desktop\semestr4\analiza sygnalow\
    raport 2\melody.with.noise.wav'

// Wczytanie plikow WAV
[audioSignal_clear, fsc] = wavread(filename_clear)
```

```
[audioSignal_noisy, fsn] = wavread(filename_noisy)

// Wczytujemy melodie z Arduino
filename_recorded = 'C:\Users\gusia\Desktop\semestr4\analiza
    sygnalow\raport 2\melody.arduino.wav'
[audioSignal_recorded, fsa] = wavread(filename_recorded)

// Przyciecie pierwszych 2.85 sekund z nagranego sygnalu audio
cut_samples = round(2.85 * fsa)
audioSignal_recorded = audioSignal_recorded(cut_samples + 1:$)

//Rysowanie wykresow
subplot(3, 1, 1)
plot(audioSignal_clear)
title("Czysty sygnal")
h = gca()
h.data_bounds = [0,-2;1000,2]

subplot(3, 1, 2)
plot(audioSignal_noisy)
title("Sygnal z szumem")
h = gca()
h.data_bounds = [0,-2;1000,2]


subplot(3, 1, 3)
plot(audioSignal_recorded)
title("Nagrany sygnal")
h = gca()
h.data_bounds = [0,-0.003;1000,0.003]
```

Aby porównać i przedstawić spektogramy dla trzech dźwięków skorzystamy z wbudowanej funkcji 'mapsound' z Scialaba.



Rysunek 4: Porównanie spektogramów dla 3 sygałów.

Fragment kodu z pliku ' spektogram.sci'

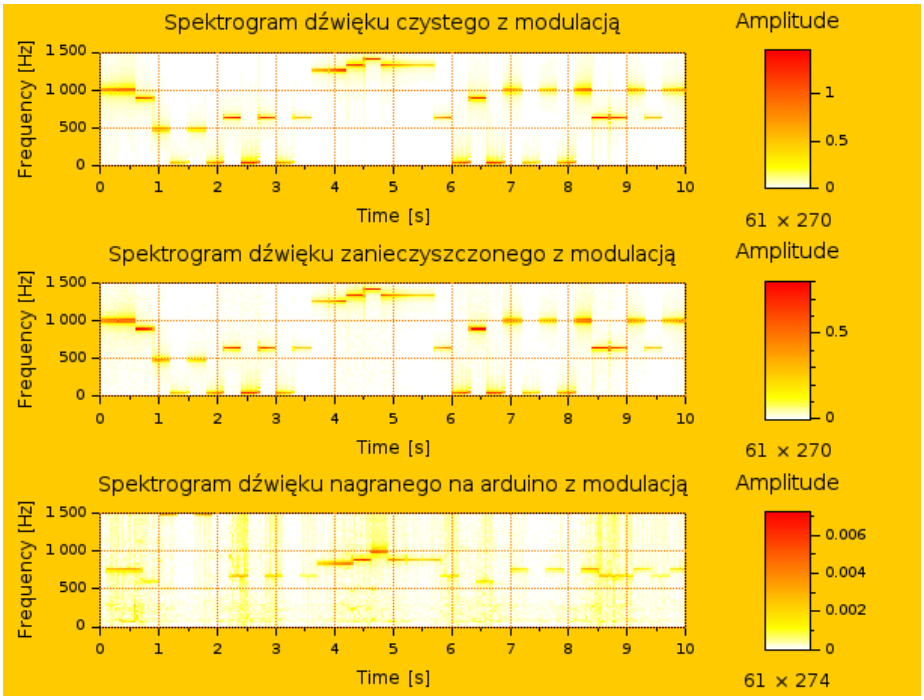
```
1
2  // Generowanie spektrogramow przy uzyciu funkcji mapsound
3  figure
4  subplot(3, 1, 1)
5  mapsound(audioSignal_clear, 0.04, 1500, fsc, autumncolormap)
6  title('Spektrogram dzwieku czystego')
7  h = gca()
8  h.data_bounds = [0, 0; 10, 1500]
9
10 subplot(3, 1, 2)
11 mapsound(audioSignal_noisy, 0.04, 1500, fsn, autumncolormap)
12 title('Spektrogram dzwieku zanieczyszczonego')
13 h = gca()
14 h.data_bounds = [0, 0; 10, 1500]
15
16 subplot(3, 1, 3)
17 mapsound(audioSignal_recorded, 0.04, 1500, fsa, autumncolormap)
18 title('Spektrogram dzwieku nagranego na arduino')
19 h = gca()
20 h.data_bounds = [0, 0; 10, 1500]
```

5.1 Modulacja amplitudy

Modulacja amplitudy w czasie polega na zmienianiu amplitudy sygnału nośnego w zależności od amplitudy sygnału modulującego. W naszym kodzie dokonujemy tego poprzez mnożenie sygnału nośnego przez sygnał modulujący. Uzyskany w wyniku sygnał zmodulowany jest sygnałem wąskopasmowym, który nadaje się np. do transmisji drogą radiową. Oto jak to realizujemy:

1. **Generowanie sygnału nośnego:**
    - Sygnał nośny to nasz podstawowy sygnał audio, który chcemy modulować. W naszym przypadku jest to sygnał odczytany z pliku WAV.
2. **Generowanie sygnału modulującego:**
    - Sygnał modulujący jest generowany jako wolno zmieniająca się fala sinusoidalna. Ma niższą częstotliwość niż sygnał nośny, co sprawia, że jego oscylacje są wolniejsze. Dzięki temu może zmieniać amplitudę sygnału nośnego w czasie.
3. **Modulacja amplitudy:**
    - Modulację amplitudy realizujemy przez mnożenie próbek sygnału nośnego przez odpowiednie próbki sygnału modulującego. W ten sposób amplituda sygnału nośnego zostaje zmieniona zgodnie z wartością sygnału modulującego.

Aby porównać i przedstawić spektogramy dla trzech dźwięków skorzystamy z wbudowanej funkcji 'mapsound' z Scialaba.



Rysunek 5: Porównanie spektogramów dla 3 sygałów z modulacją amplitudy.

Fragment kodu z pliku 'spektogram.modulacja.sci'

```
// Funkcja generujaca sygnal modulujacy
function modSignal = generateModulatingSignal(duration,
    samplingRate, modFreq)
    t = (0:1/samplingRate:duration/1000)' // Wektor czasu
    modSignal = 1 + 0.5 * sin(2 * %pi * modFreq * t) // Sygnal
    modulujacy
endfunction

[audioSignal_clear, fsc] = wavread(filename_clear)
[audioSignal_noisy, fsn] = wavread(filename_noisy)
[audioSignal_recorded, fsa] = wavread(filename_recorded)

// Generowanie sygnalow z modulacja amplitudy
modFreq = 0.5 // Czestotliwosc modulacji

modSignal_clear = generateModulatingSignal(length(audioSignal_clear
    ) / fsc * 1000, fsc, modFreq)
modSignal_noisy = generateModulatingSignal(length(audioSignal_noisy
    ) / fsn * 1000, fsn, modFreq)
modSignal_recorded = generateModulatingSignal(length(
```

```
    audioSignal_recorded) / fsa * 1000, fsa, modFreq)

// Dopasowanie dlugosci sygnalow modulujacych
modSignal_clear = modSignal_clear(1:length(audioSignal_clear))
modSignal_noisy = modSignal_noisy(1:length(audioSignal_noisy))
modSignal_recorded = modSignal_recorded(1:length(
    audioSignal_recorded))

// Modulacja sygnalow
modulatedSignal_clear = audioSignal_clear .* modSignal_clear
modulatedSignal_noisy = audioSignal_noisy .* modSignal_noisy
modulatedSignal_recorded = audioSignal_recorded .*
    modSignal_recorded

// Generowanie spektrogramow przy uzyciu funkcji mapsound
figure
subplot(3, 1, 1)
mapsound(modulatedSignal_clear, 0.04, 1500, fsc, autumncolormap)
title('Spektrogram dzwieku czystego z modulacja')
h = gca()
h.data_bounds = [0, 0; 10, 1500]

subplot(3, 1, 2)
mapsound(modulatedSignal_noisy, 0.04, 1500, fsn, autumncolormap)
title('Spektrogram dzwieku zanieczyszczonego z modulacja')
h = gca()
h.data_bounds = [0, 0; 10, 1500]

subplot(3, 1, 3)
mapsound(modulatedSignal_recorded, 0.04, 1500, fsa, autumncolormap)
title('Spektrogram dzwieku nagranego na arduino z modulacja')
h = gca()
h.data_bounds = [0, 0; 10, 1500]
```

5.2 Wnioski

Porównanie spektrogramów

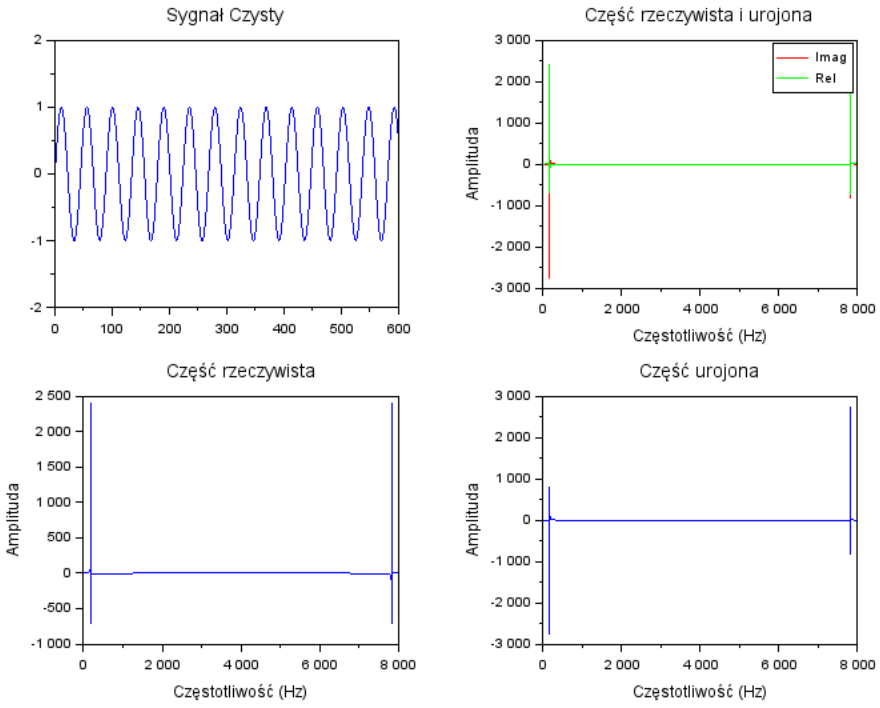
Spektrogramy trzech badanych dźwięków (czystego, z szumem i nagranego za pomocą Arduino) wykazują znaczną zbieżność. Oznacza to, że pomimo różnic w źródle i charakterystyce dźwięków, ich rozkład częstotliwościowy w czasie jest podobny. Zbieżność ta sugeruje, że podstawowe cechy akustyczne analizowanych sygnałów są zachowane niezależnie od obecności szumu czy metody nagrania.

Wpływ modulacji amplitudy w czasie na spektrogramy

Analiza modulacji amplitudy w czasie dla każdego z trzech sygnałów wykazała, że nie wpływa ona znacząco na ich spektrogramy. Oznacza to, że choć modulacja amplitudy zmienia intensywność sygnału w czasie, to podstawowy rozkład częstotliwościowy pozostaje niezmieniony. Można zatem stwierdzić, że struktura częstotliwościowa badanych sygnałów jest stabilna i niezależna od modulacji amplitudy.

6 Porównanie transformaty Fouriera dla trzech sygnałów

6.1 Transformata Fouriera dla sygnału czystego



Rysunek 6: Transformata Fouriera dla sygnału czystego.

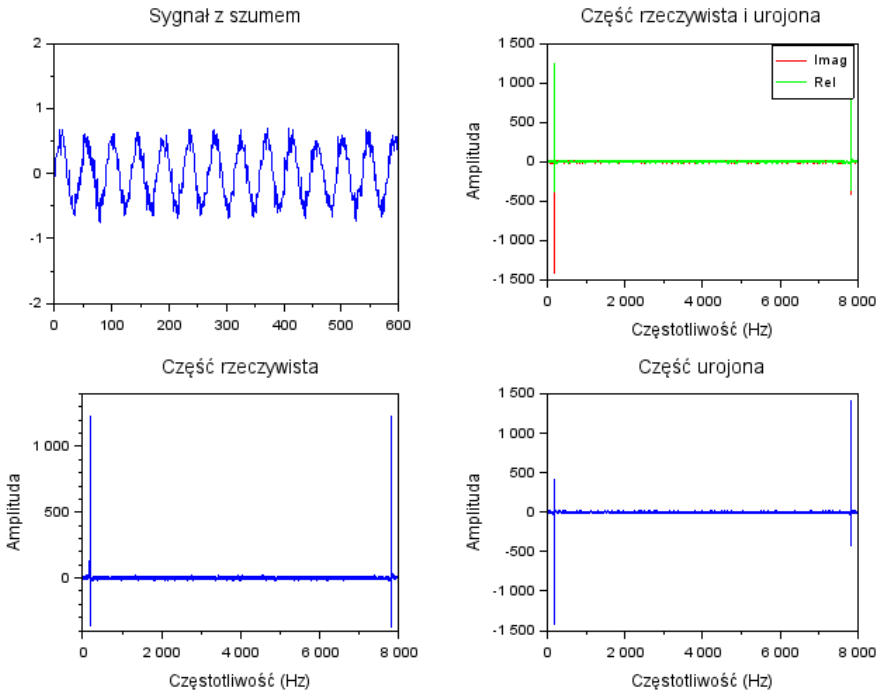
Fragment kodu z pliku ' DFT.1.sygnal.sci'

```
    // Definicja funkcji DFT
function Xk = DFT_fun(xn)
    N = length(xn)
    k = [0:N-1]'
    n = [0:N-1]
    W = exp(-%i * 2 * %pi / N * (k * n))
    Xk = W * xn
endfunction


// Obliczenie DFT dla sygnalu czystego
DFT_clear = DFT_fun(audioSignal_clear')

```

```
// Wykres dla sygnalu clear

subplot(2, 2, 1)
plot(audioSignal_clear)
title("Sygnal Czysty")
h = gca()
h.data_bounds = [0,-2;600,2]

subplot(2,2,2)
plot(imag(DFT_clear),'r')
plot(real(DFT_clear),'g')
title('Czesc rzeczywista i urojona')
xlabel('Czestotliwosc (Hz)')
ylabel('Amplituda')
legend("Imag", "Rel")


subplot(2, 2, 3)
plot(real(DFT_clear))
title('Czesc rzeczywista')
xlabel('Czestotliwosc (Hz)')
ylabel('Amplituda')

subplot(2, 2, 4)
plot(imag(DFT_clear))
title('Czesc urojona ')
xlabel('Czestotliwosc (Hz)')
ylabel('Amplituda')
```

6.2 Transformata Fouriera dla sygnału z szumem

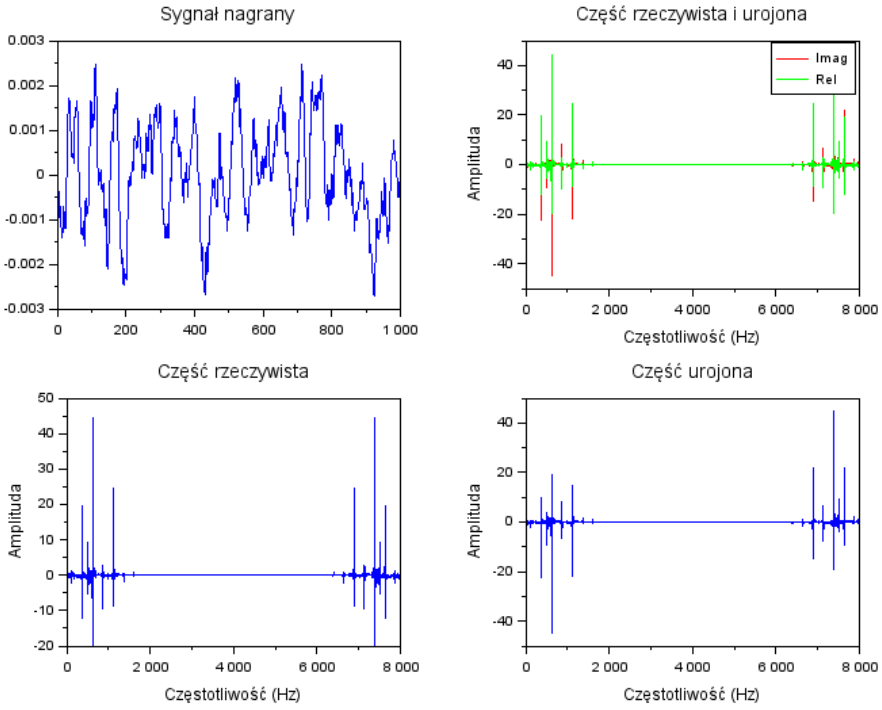
Kod do generowania poniższych wykresów jest bardzo podobny do kodu generującego wykresy z rysunku 6. Kod w pliku ' DFT.2.sygnal.sci'



Rysunek 7: Transformata Fouriera dla sygnału z szumem.

6.3 Transformata Fouriera dla sygnału nagranego

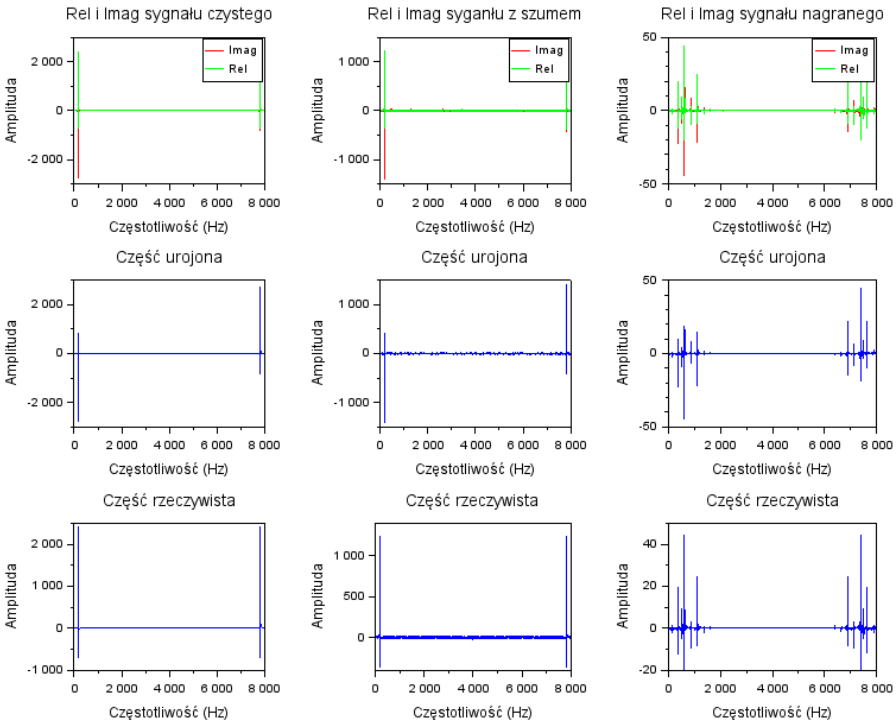
Kod do generowania poniższych wykresów jest bardzo podobny do kodu generującego wykresy z rysunku 6. Kod w pliku ' DFT.3.sygnal.sci'



Rysunek 8: Transformata Fouriera dla sygnału nagranego.

6.4 Porównanie

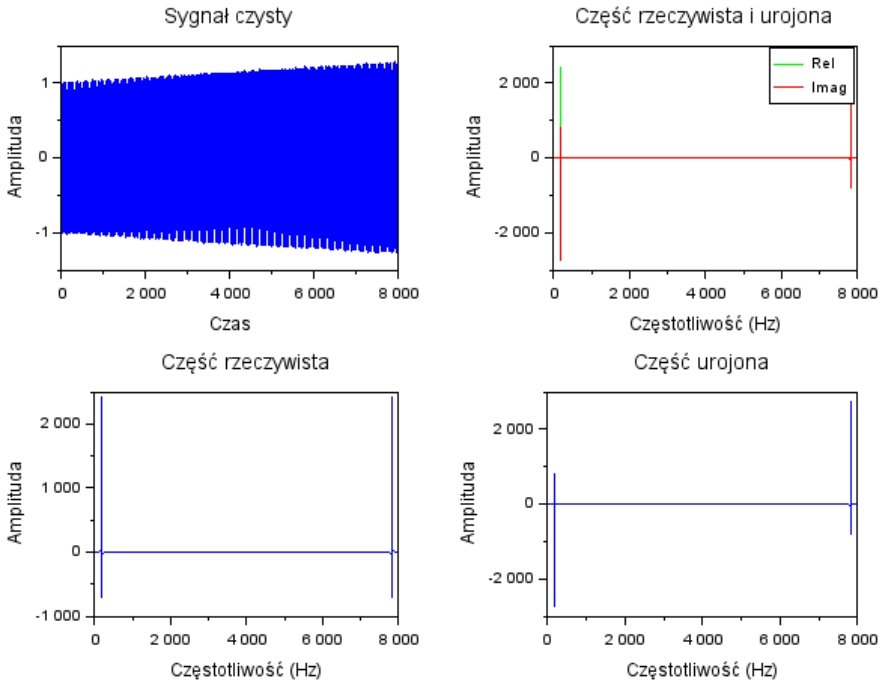
Poniżej przedstawiono porównanie transformt Fouriera dla trzech dźwięków: czysty dźwięk wygenerowany w Scilabie, ten sam sygnał z szumem oraz dźwięk wygenerowany przez Arduino i zarejestrowany przez mikrofon. Kod w pliku 'DFT.porownanie.sci'



Rysunek 9: Porównanie transformaty Fouriera dla trzech sygnałów.

6.5 Modulacja amplitudy

6.5.1 Modulacja amplitudy dla sygnału czystego.



Rysunek 10: Modulacja amplitudy dla sygnału czystego.

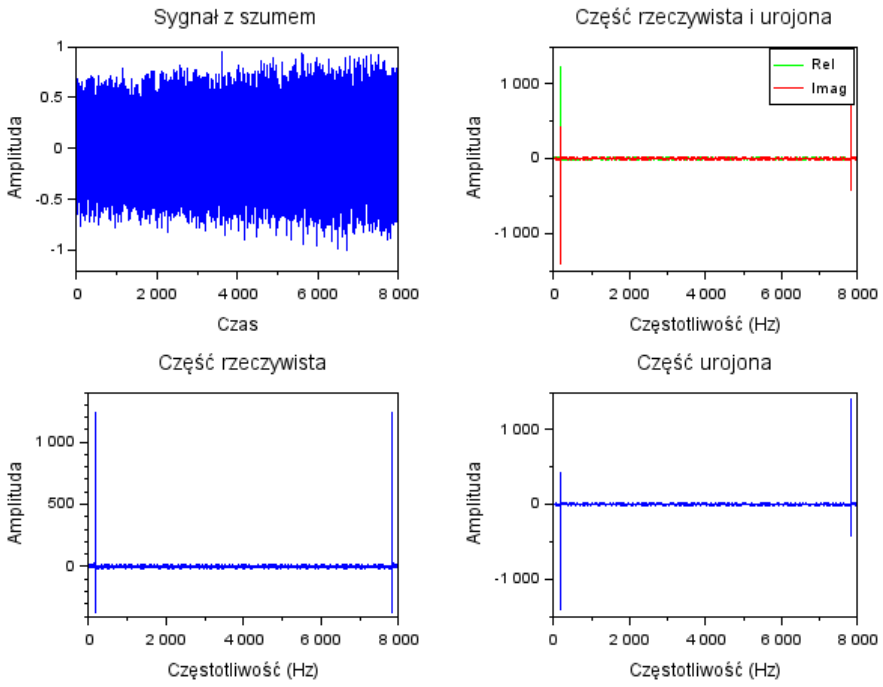
Fragment kodu z pliku ' DFT.mod.1.sci'.

```
1
2  // Funkcja generujaca sygnal modulujacy
3  function modSignal = generateModulatingSignal(duration,
       samplingRate, modFreq)
4      t = (0:1/samplingRate:duration/1000)' // Wektor czasu
5      modSignal = 1 + 0.5 * sin(2 * %pi * modFreq * t) // Sygnal
       modulujacy
6  endfunction
7
8  // Upewnijmy sie, ze sygnaly sa wektorami kolumnowymi
9  audioSignal_clear = audioSignal_clear(:)
10 audioSignal_noisy = audioSignal_noisy(:)
11 audioSignal_recorded = audioSignal_recorded(:)
12
13 // Obliczenie DFT dla kazdego sygnalu
```

```
DFT_clear = DFT(audioSignal_clear)
DFT_noisy = DFT(audioSignal_noisy)
DFT_recorded = DFT(audioSignal_recorded)

// Generowanie sygnalow z modulacja amplitudy
modFreq = 0.5; // Czestotliwosc modulacji
modSignal_clear = generateModulatingSignal(num_samples / fsc *
    1000, fsc, modFreq)
modSignal_noisy = generateModulatingSignal(num_samples / fsn *
    1000, fsn, modFreq)
modSignal_recorded = generateModulatingSignal(num_samples / fsa *
    1000, fsa, modFreq)

// Upewnijmy sie, ze sygnaly modulujace sa wektorami kolumnowymi o
    odpowiednich wymiarach
modSignal_clear = modSignal_clear(1:num_samples)
modSignal_noisy = modSignal_noisy(1:num_samples)
modSignal_recorded = modSignal_recorded(1:num_samples)

// Modulacja sygnalow
modulatedSignal_clear = audioSignal_clear .* modSignal_clear
modulatedSignal_noisy = audioSignal_noisy .* modSignal_noisy
modulatedSignal_recorded = audioSignal_recorded .*
    modSignal_recorded

// Dodanie szumu do sygnalow
noise_clear = rand(length(modulatedSignal_clear), 1, "normal") *
    0.01
noise_noisy = rand(length(modulatedSignal_noisy), 1, "normal") *
    0.01
noise_recorded = rand(length(modulatedSignal_recorded), 1, "normal"
    ) * 0.01

modulatedSignal_clear = modulatedSignal_clear + noise_clear
modulatedSignal_noisy = modulatedSignal_noisy + noise_noisy
modulatedSignal_recorded = modulatedSignal_recorded +
    noise_recorded

// Wykresy dla czystego sygnalu

subplot(2, 2, 1)
plot(modulatedSignal_clear)
title('Sygnal czysty')
xlabel('Czas')
ylabel('Amplituda')

subplot(2, 2, 2)
plot(real(DFT_clear), 'g') // Rzeczywista na zielono
plot(imag(DFT_clear), 'r') // Urojona na czerwono
title('Czesc rzeczywista i urojona')
xlabel('Czestotliwosc (Hz)')
ylabel('Amplituda')
legend('Rel', 'Imag')

subplot(2, 2, 3)
plot(real(DFT_clear))
title('Czesc rzeczywista')
```

```
xlabel('Czestotliwosc (Hz)')
ylabel('Amplituda')

subplot(2, 2, 4)
plot(imag(DFT_clear))
title('Czesc urojona')
xlabel('Czestotliwosc (Hz)')
ylabel('Amplituda')
```

6.5.2 Modulacja amplitudy dla sygnału z szumem.

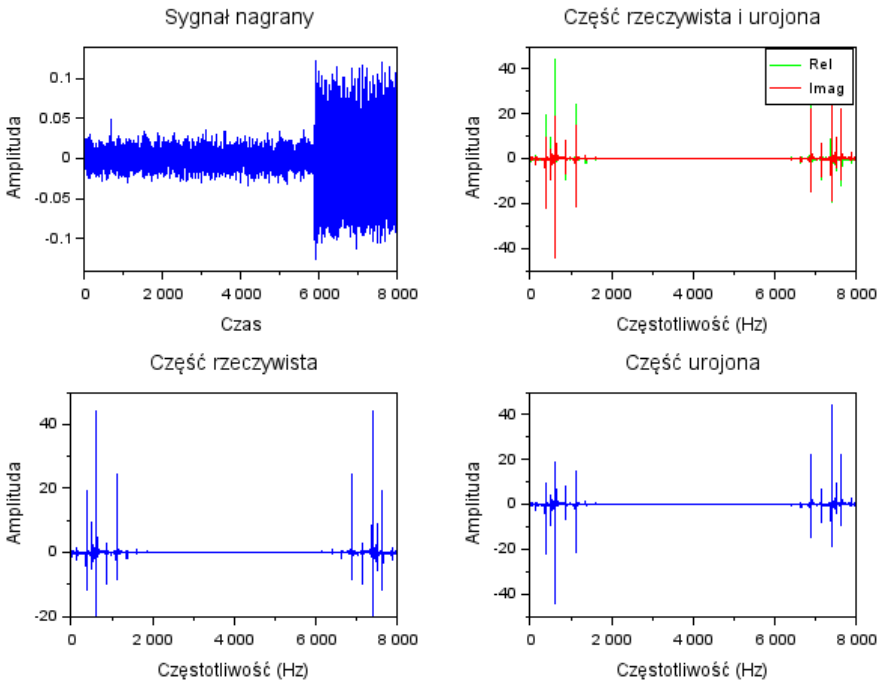


Rysunek 11: Modulacja amplitudy dla sygnału z szumem.

Kod do generowania powyższych wykresów jest bardzo podobny do kodu generującego wykresy z rysunku 10. Kod w pliku ' DFT.mod.2.sci'

6.5.3 Modulacja amplitudy dla sygnału nagranego.

Kod do generowania poniższych wykresów jest bardzo podobny do kodu generującego wykresy z rysunku 10. Kod w pliku ' DFT.mod.3.sci'



Rysunek 12: Modulacja amplitudy dla sygnału nagranego.

6.6 Wnioski

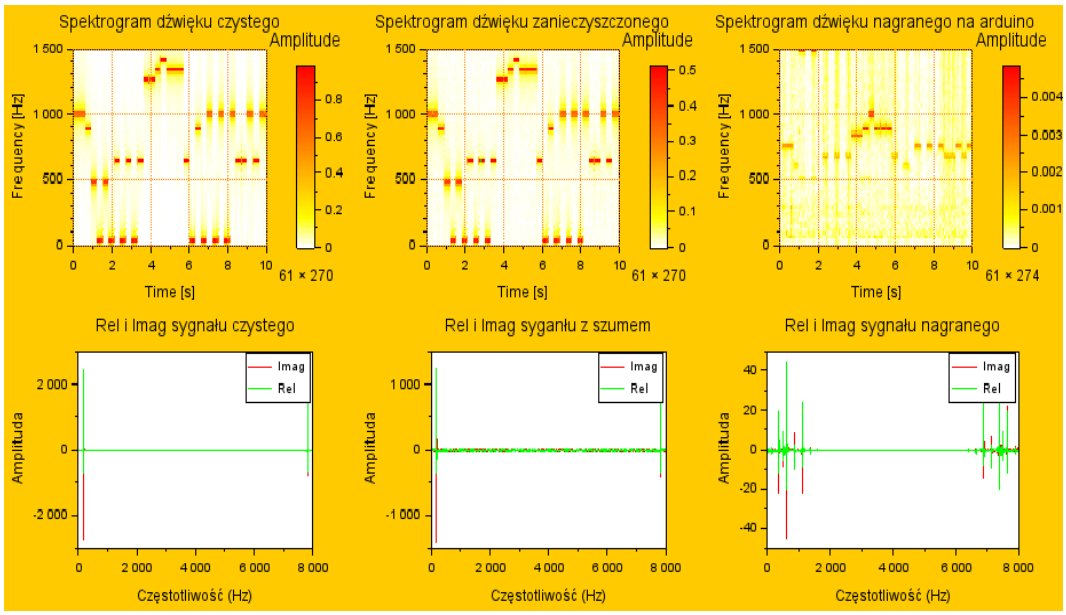
Analiza transformaty Fouriera dla trzech rodzajów sygnałów: czystego, zanieczyszczonego (z szumem) oraz nagranego, pozwala na wyciągnięcie następujących wniosków:

- **Czysty sygnał:** Widmo częstotliwościowe jest dobrze zdefiniowane z wyraźnymi pikami przy częstotliwościach odpowiadających nutom w melodii.
- **Sygnał z szumem:** Widmo częstotliwościowe jest podobne do czystego sygnału. Różnice powstają w zakłóceniach (małych pikach pośrodku w miejscach, gdzie w czystym sygnale widmo częstotliwościowe nie wykazuje żadnych odchyleń.
- **Sygnał nagrany:** Widmo częstotliwościowe jest bardziej rozmyte i zawiera dodatkowe składowe częstotliwościowe. Piki są przesunięte i mniej wyraźne z powodu zakłóceń wprowadzonych podczas nagrywania.
- **Modulacja:** Modulacja amplitudy sygnału w czasie nie wpływa znacząco na wykresy transformaty Fouriera, różnice są niewielkie. Transformata Fouriera skutecznie identyfikuje obecność szumów i zakłóceń oraz analizuje strukturę częstotliwościową sygnałów.

Podsumowując, czysty sygnał ma najbardziej wyraźne widmo częstotliwościowe, podczas gdy sygnał nagrany maja najbardziej złożone widmo z mniej wyraźnymi pikami. Modulacja amplitudy w czasie nie wprowadza znaczących różnic w analizie widma częstotliwościowego.

7 Porównanie spektogramów i transformaty Fouriera dla trzech sygnałów

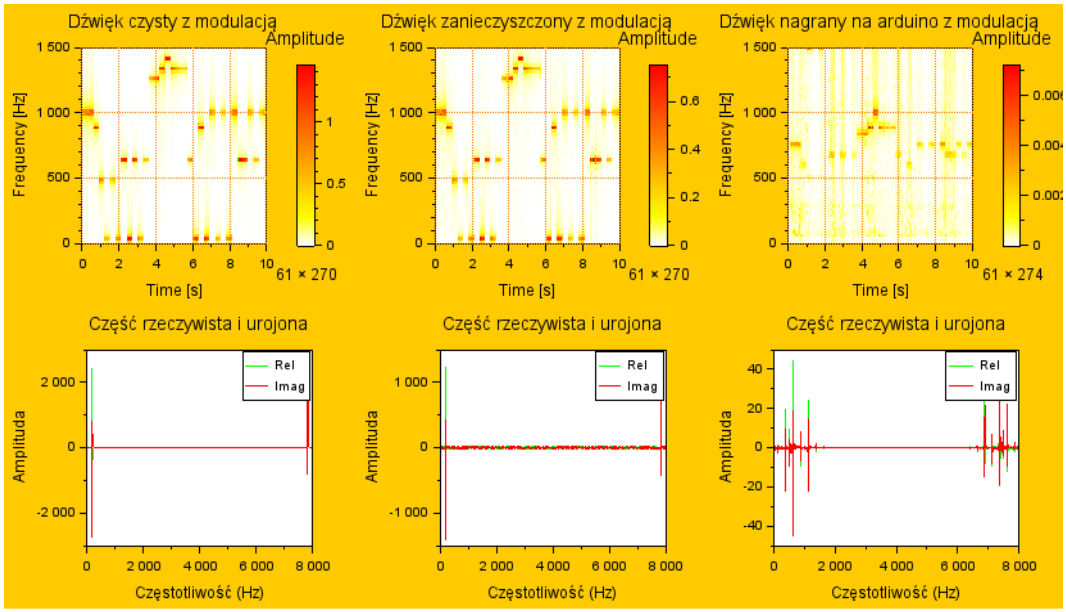
Dla podsumowania możemy przedstawić wykresy spektogramów oraz wykresy transformat Fouriera dla odpowiednich dźwięków na jednym rysunku. Kod w pliku ' spek.DFT.sci'



Rysunek 13: Spektogramy i transformaty Fouriera dla trzech sygnałów.

7.1 Modulacja amplitudy

Dla podsumowania możemy również przedstawić wykresy spektogramów oraz wykresy transformat Fouriera dla odpowiednich dźwięków z modulacją amplitudy na jednym rysunku. Kod w pliku ' spek.DFT.mod.sci'



Rysunek 14: Spektogramy i transformaty Fouriera dla trzech sygnałów z modulacją amplitudy.