VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
DUOMENŲ MOKSLO IR SKAITMENINIŲ TECHNOLOGIJŲ INSTITUTAS

GARSO SIGNALŲ APDOROJIMAS

Antrasis laboratorinis darbas

Garso signalų analizė laiko srityje

Darbą atliko 4 kurso ISI 2 grupės studentas

Tomas Šinkūnas

Vilnius, 2023

Įvadas

Darbo tikslas

Garso signalų analizė laiko srityje, analizės pritaikymas signalo segmentams aptikti.

Darbo užduotis

Sukurti priemonę garso failams nuskaityti, nuskaitytiesiems signalams (ar jų atkarpoms) grafiškai atvaizduoti. Įvertinti signalo energijos bei nulio kirtimų skaičiaus (NKS) kitimą laike. Pritaikius slenksčio principą pabandyti aptikti signalo atkarpas, segmentus ar kitus vienetus įrašuose.

Darbo priemonės

Darbui atlikti buvo naudojama *Python* programavimo kalba su *matplotlib*, *numpy* ir *wave* bibliotekomis.

Duomenys

Šiame laboratoriniame darbe yra analizuojami 2 garso failai:

1. Applause.wav
2. Opera-vocal_129bpm_F_minor.wav (toliau bus vadinamas kaip opera.wav)

Darbo eiga

Techninės charakteristikos

Garso failų techninei analizei buvo pasitelkta *python* kalbos *wave* biblioteka. Nuskaičius failus ir išsaugojus duomenis į masyvą tolimesniam apdirbimui, gauti tokie techniniai duomenys apie kiekviena failą. Rezultatai pateikti lentelėje nr.1.

Lentelė nr.1. Failų techninės charakteristikos

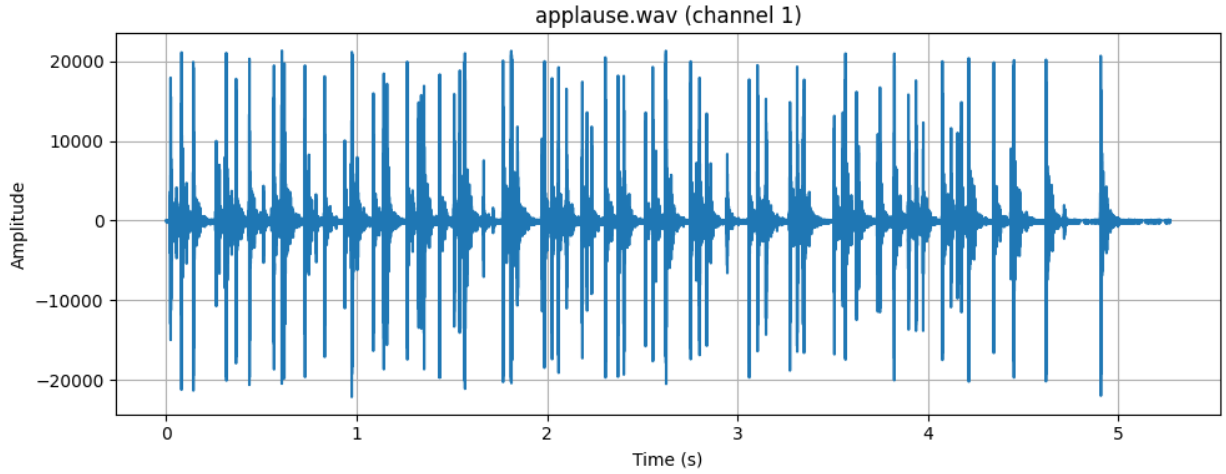
	Applause.wav	Opera.wav
bitRate (bits per second)	705600	705600
bitsPerSample (bits)	16	16
compressionType	NONE	NONE
Duration (seconds)	5.27	9.69
fileSizeComputed (bytes)	465072.0	855272.0
numChannels	1	1

numSamples	232536	427636
sampleRate (Hz)	44100	44100

Laiko diagrama

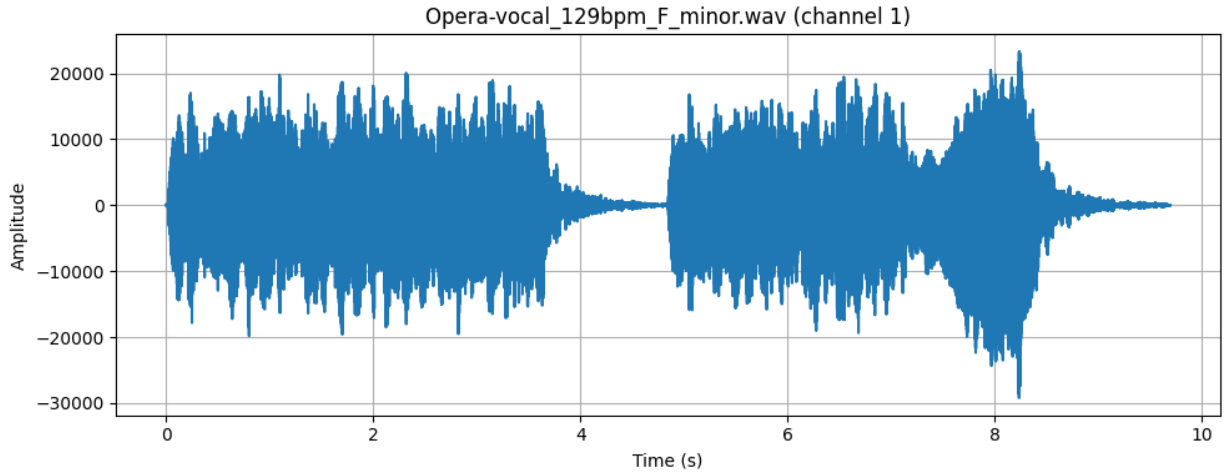
Garso failo reikšmės šiame darbe nebuvo normalizuojamos, t.y. laiko diagramos vertikalioji ašis atitinka realias signo amplitudės reikšmes.

Failo *applause.wav* laiko diagrama pateikta paveiksle nr. 1. Šio garsinio failo trukmė nesiekia 6 sekundžių, ir įrašas atrodo gana monotoniškas, pasikartojantis, su staigiais amplitudžių svyravimais.



1 pav. Failo Applause.wav laiko diagrama

Failo *opera.wav* laiko diagrama pateikta paveiksle nr. 2. Jo trukmė - iki 10 skundžių ir amplitudinės charakteristikos per daug neišsiskiria. Iš diagramos galima spėti, jog įraše yra pasakomos kelios frazės ar žodžiai, su pauze tarp jų ties 5'a sekunde.

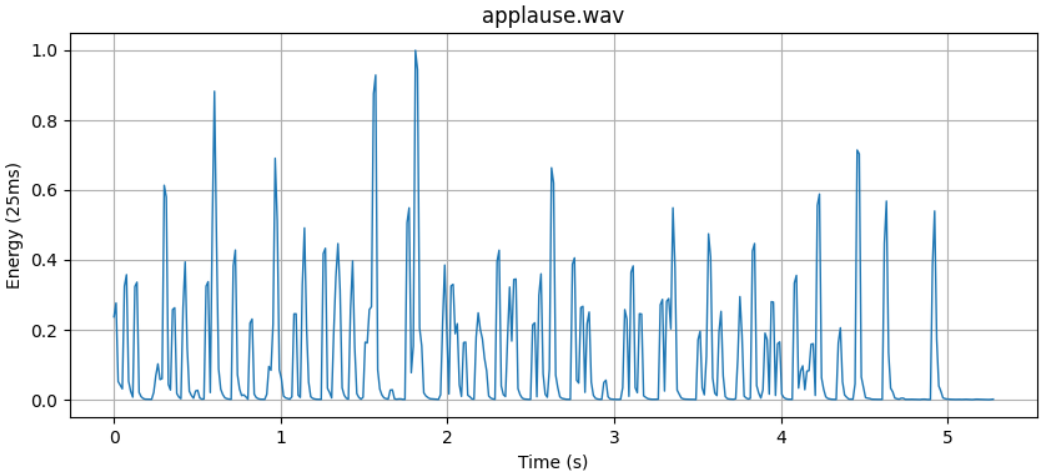


2 pav. Failo Opera.wav laiko diagrama

Energijos diagrama

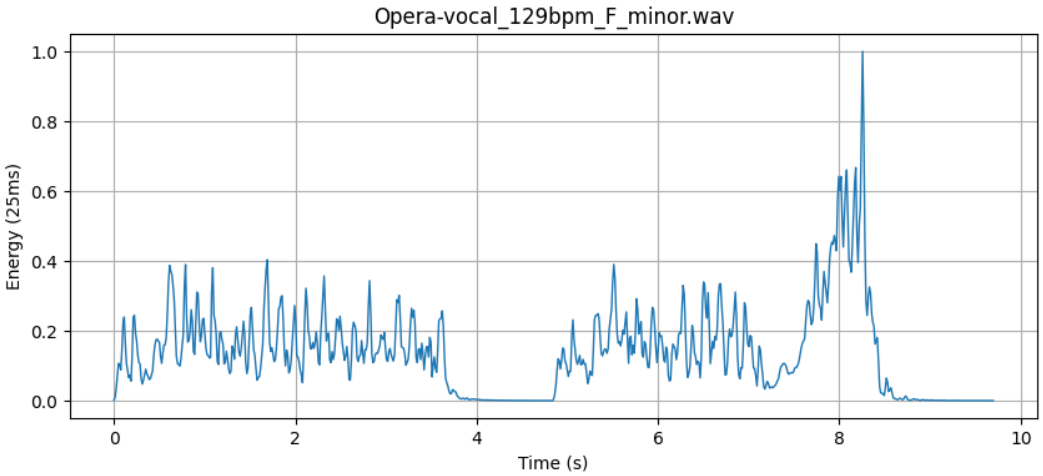
Skaičiuojant signalo energiją buvo pasirinktas 25ms kadro ilgis.

Kaip matome iš *Applause.wav* energijos diagramos (paveikslas nr. 3), garsas turėjo pastovius energijos svyravimus ir jo energija gana monotoniškai kinta – didžiausia energija yra matoma ties 2'a sekunde.



3 pav. Failo Applause.wav energijos diagrama

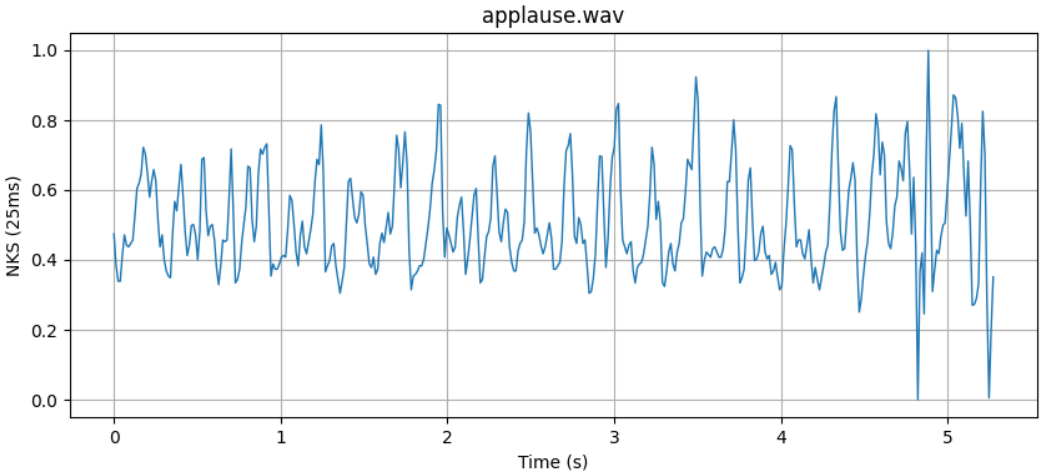
Tuo metu 4 pav. Opera.wav energijos diagrama parodo, jog šio garso energija yra išlaikyta daugiau mažiau viedonai, o ties failo pabaiga gerokai padidėja. Taipogi galime pastebėti, kad energija visiškai prarandama ties failo viduriu (apie 5'ą seundę) - tai dar kartą patvirtina, jog tarp žodžių (ar frazių) buvo padaryta pauzė.



4 pav. Failo Opera.wav energijos diagrama

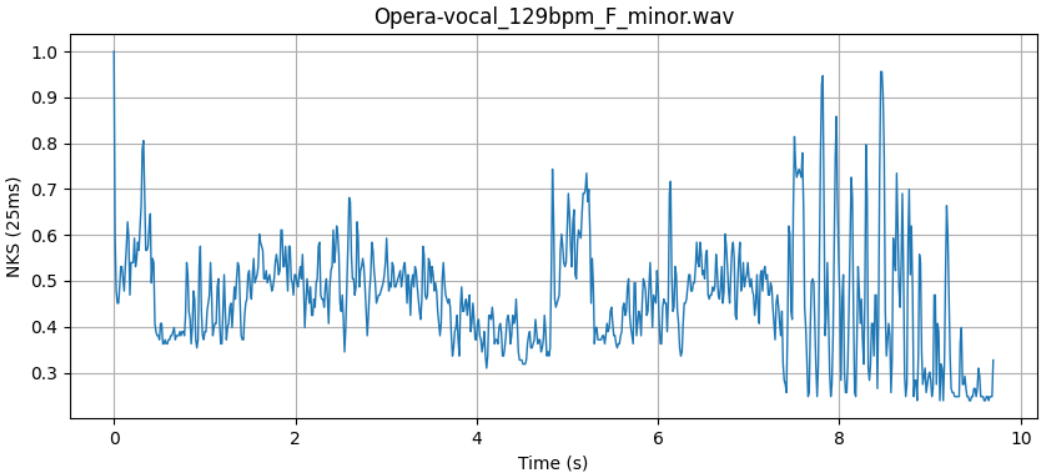
NKS diagrama

Iš *applause.wav* nulių kirtimo skaičiaus (NKS) diagramos (paveikslas 5) galima matyti, jog reikšmės yra daug maž vienodos viso įrašo metu. Tik ties failo pabaiga (apie 5’ą sekundę) matome staigius reikšmių sumažėjimus. Iš čia galime daryti prielaidą, jog plojimų garsumas buvo panašus viso įrašo metu, o jų skaičius sumažėjo tik ties pabaiga.



5 pav. Failo Applause.wav NKS diagrama

Opera.wav NKS diagrama pateikta paveiksle nr. 6. Iš čia matome, kad nulių kirtimų skaičius yra daug maž vienodas, išskyrus failo pabaigą, kur kirtimų dažnis padidėja. Pagal tai galime teiti, jog garsas viso failo metu buvo išlaikytas daug maž vienodai, išskyrus pabaigą, kur garsas padidėja.



6 pav. Failo Opera.wav NKS diagrama

Diagramų palyginimas

Laiko diagrama tiesiog vaizduoja signalą laike, kur *x ašis* yra laikas, o *y ašis* yra signalo amplitudė. Ji rodo, kaip signalas kinta nuo vieno momento iki kito, bet nesuteikia jokios papildomos informacijos apie energijos ar nulio kirtimo dažnio savybes.

Energijos diagrama vaizduoja signalo energijos pasiskirstymą laike. Ji leidžia atskirti, kur signalo energija yra aukštesnė, o tai gali padėti atpažinti akustinio signalo savybes arba ryškius jo segmentus.

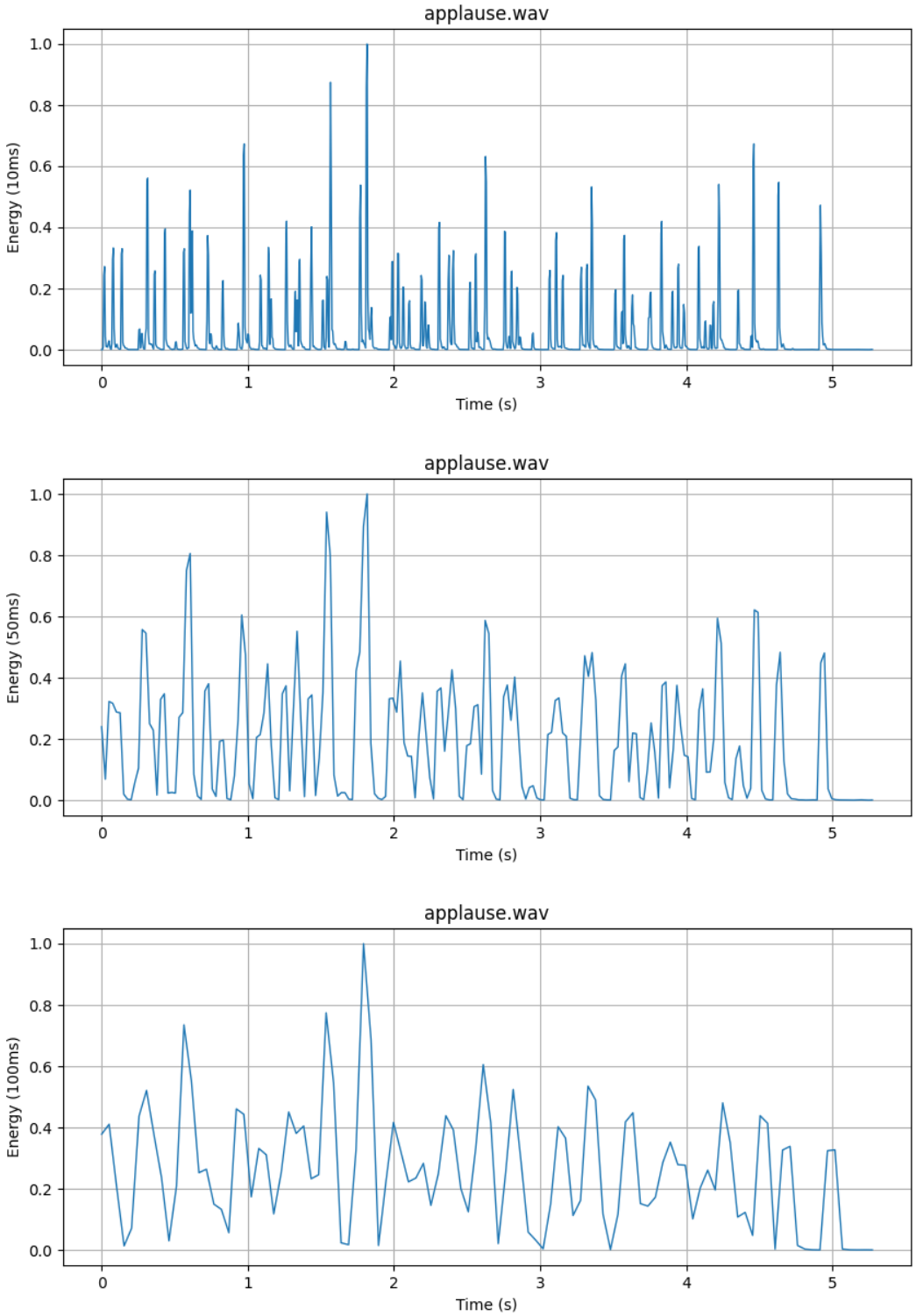
NKS diagrama rodo, kur signalo nulio kirtimo dažnis yra didžiausias. Ji padeda identifikuoti, kur signalas turi daugiau pokyčių arba yra "triukšmingesnis."

Analizuojant *Applause.wav* filą iš visų 3 diagramų matome kuriose vietose pikai - ten įvyksta plojimas, o kur reikšmės sumažėja - ten tyla.

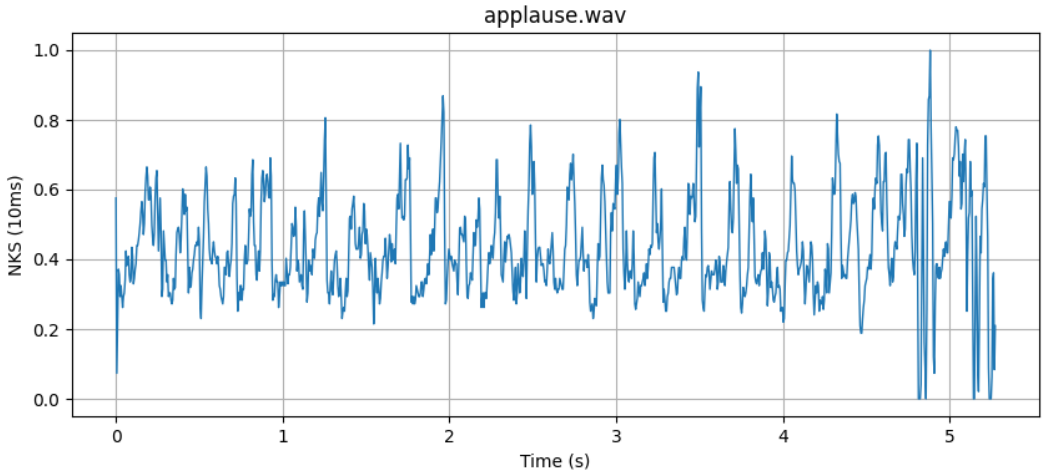
Analizuojant *Opera.wav* failą, iš laiko bei energijos diagramų matome, kurioje vietoje yra pauzė (ties 5’a sekunde) - toje vietoje tiek amplitudė, tiek energija gerokai sumažėja. Tačiau šito efekto nesimato NKS diagramoje. Kitą vertus, laiko diagrame neatspindi signalo įspūdžio sustiprėjimo, kas akivaizdžiai matoma tiek energijos, tiek NKS diagramos ties failo pabaiga (apie 8’ą sekundę).

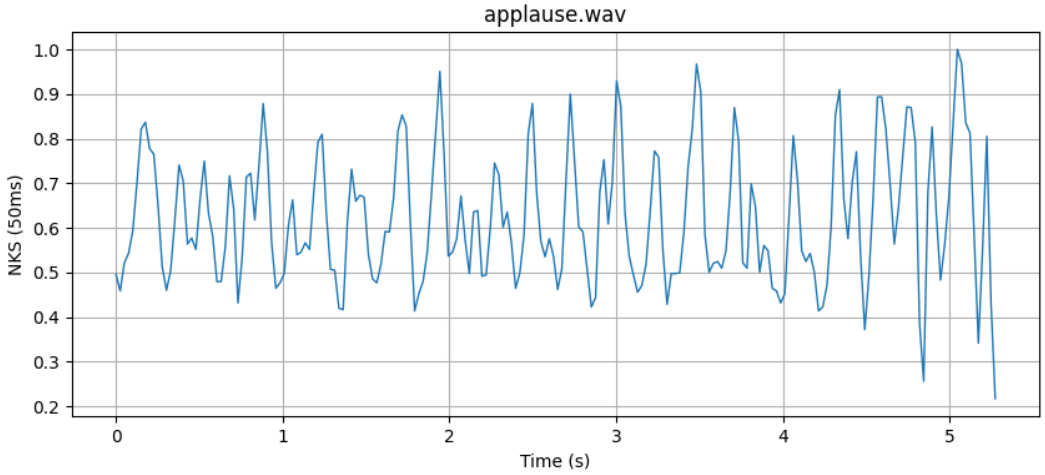
Žemiau pateiktuose paveiksluose (6, 7, 8, 9) matome, kaip rezultatai priklauso nuo kadro dydžio. Mažinant kadro dydį energija ir NKS skaičiuojama per trumpesnius laiko intervalus, kas padeda geriau užfiksuojant greitus signalo pokyčius. Tačiau to pasekoje gali nepakankamai gerai užfiksuoti žemo dažnio signalus.

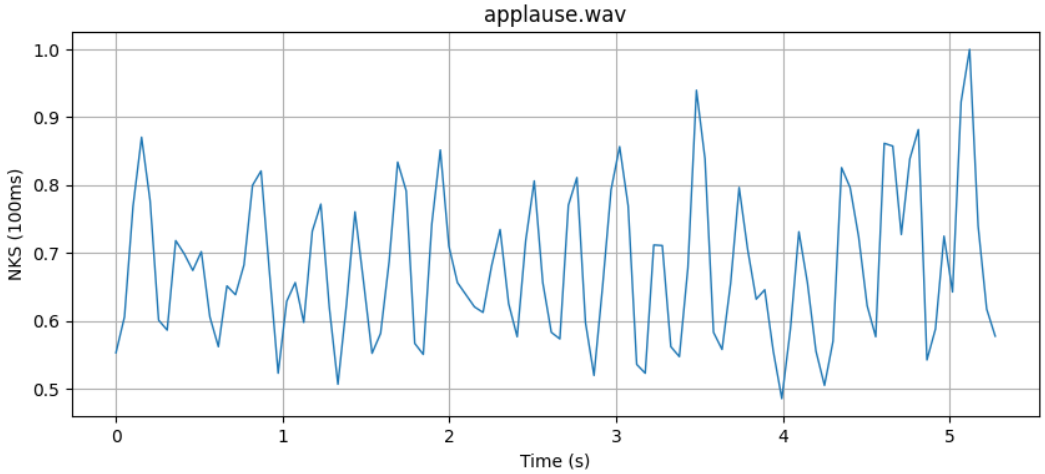
Didinant kadro dydį energija ir NKS yra skaičiuojami per ilgesnius laiko intervalus, kas padeda sušvelninti greitus signalo pokyčius. Kitą vertus, tai padeda geriau užfiksuoti žemo dažnio signalus.



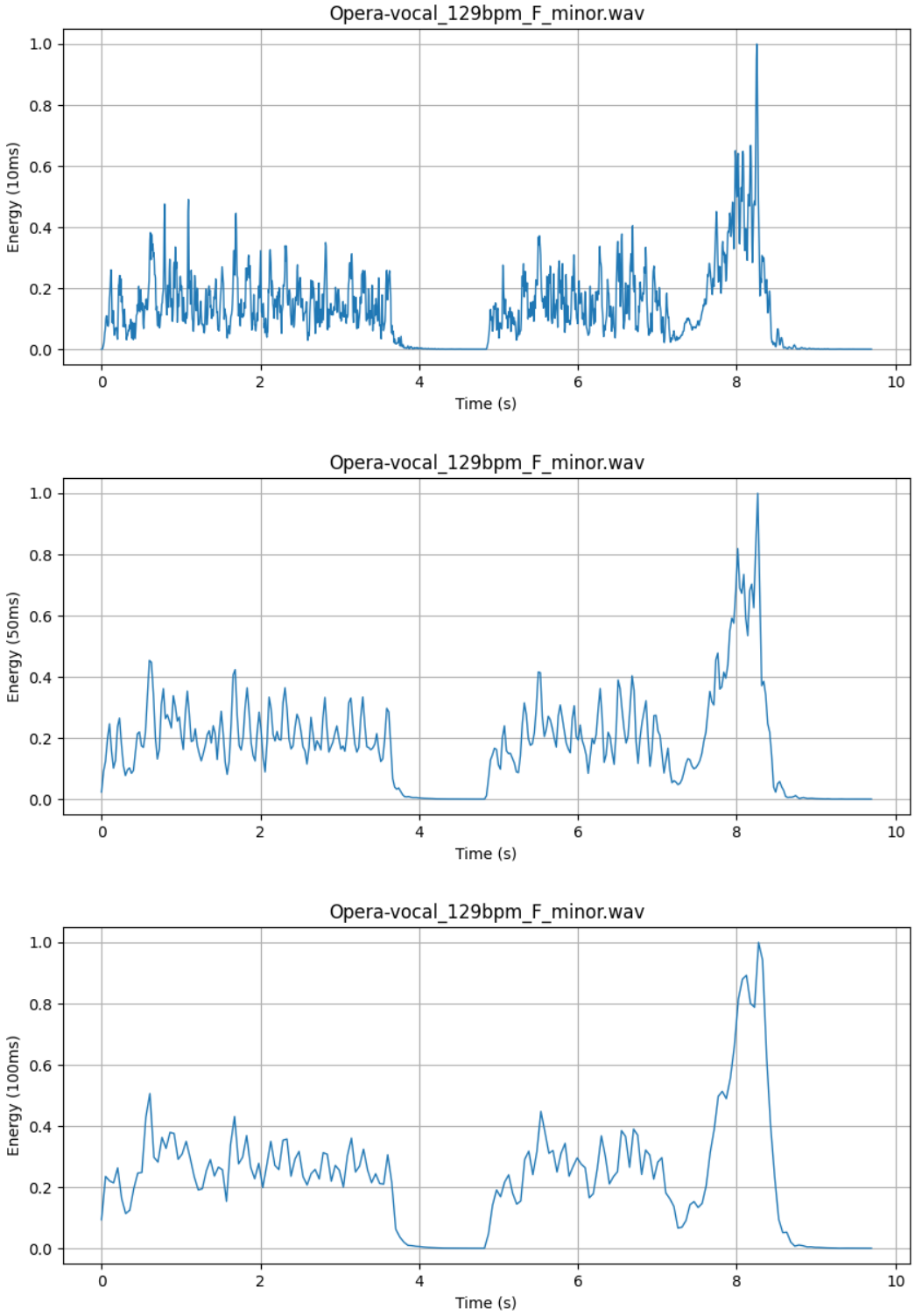
6 pav. Failo Applause.wav energijos diagrama, su kadro dydžiais 10, 50, 100



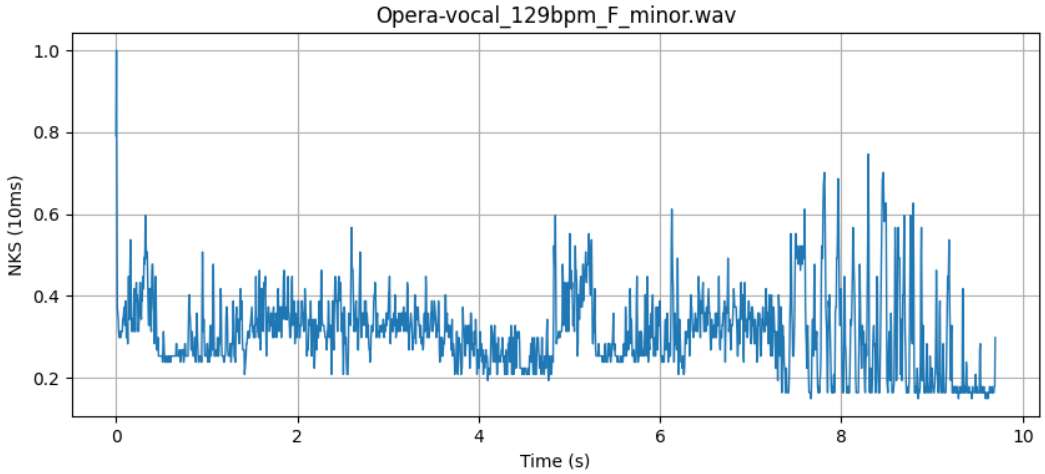


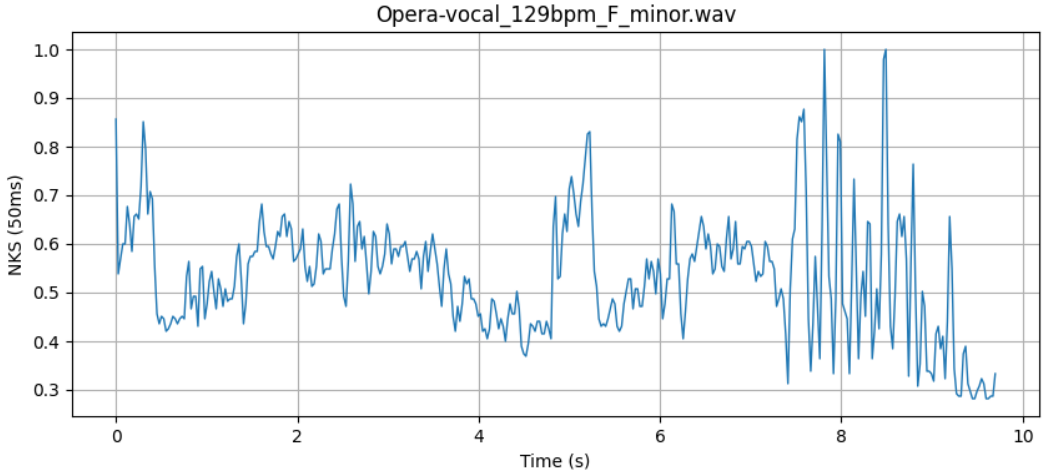


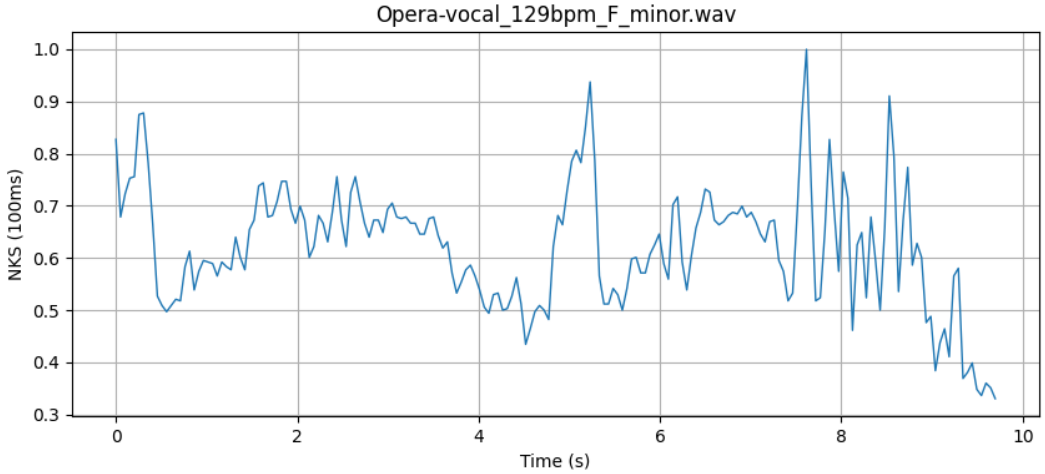
7 pav. Failo Applause.wav NKS diagrama, su kadro dydžiais 10, 50, 100



8 pav. Failo Opera.wav energijos diagrama, su kadro dydžiais 10, 50, 100





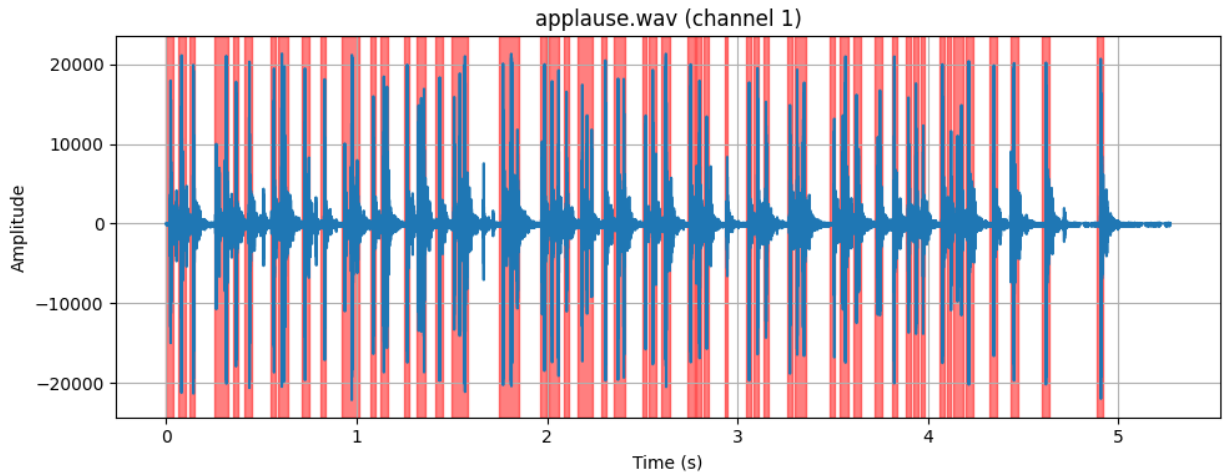


9 pav. Failo Opera.wav NKS diagrama, su kadro dydžiais 10, 50, 100

Signalo atkarpos aptikimas

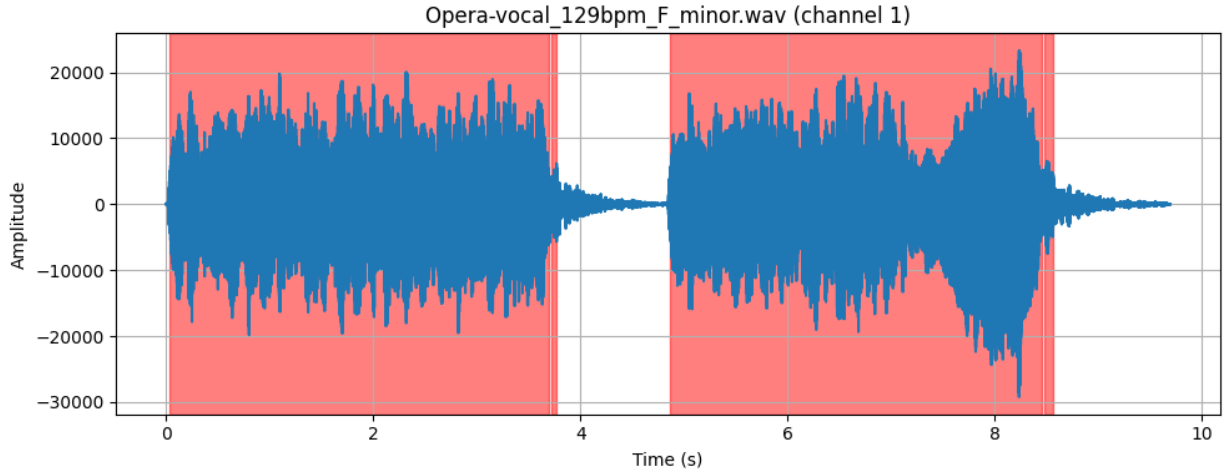
Pritaikant slenksčio metodą, signalą galima suskirstyti į atskiras atkarpas.

Pavyzdžiui, paveiksle nr. 10 pažymėti *Applause.wav* signalai, kurių energija yra > 0,05



10 pav. Failo Applause.wav signalai, kurių energija > 0,05

O iš faile *Opera.wav*, norėdami aptikti pauzes, pažymėkime tik tas signalo reikšmes, kurių energija yra > 0.021



11 pav. Failo Opera.wav signalai, kurių energija > 0,021

Išvados

Laboratorinio darbo metu buvo sukurta priemonė garso signalų nuskaitymui bei grafiniam jų atvaizdavimui. Taip pat buvo įvertintos ir signalų laikinės, energijos, bei NKS charakteristikos bei pateikti jų grafikai. Taip. pat buvo sukurta ir priemonė signalų segmentavimui pagal pasirinktą energijos vertę.

Laiko, energijos ir NKS diagramos kiekviena suteikia unikalias žinias apie garso signalus. Laiko diagrama atvaizduoja amplitudės svyravimus, o energijos ir NKS diagramos - užfiksuoja laiko charakteristikas. Tačiau nė viena iš šių diagramų tiesiogiai neparodo dažnio turinio ar spektrinių detalių signalo, kurios dažnai reikalauja papildomų spektrinės analizės technikų.

Programos kodas

Main.py

```
from WAV import WAV

if __name__ == "__main__":
    wav = WAV("Sounds/applause.wav")
    # wav = WAV("Sounds/Opera-vocal_129bpm_F_minor.wav")

    print(wav.toString())
    wav.plotSelf()

    wav.setFrameDuration(25)

    # signalo energija
    wav.plot(wav.getEnergy(), "Energy (25ms)", wav.name)

    # NKS diagrama
    wav.plot(wav.getZCR(), "NKS (25ms)", wav.name)

    energyThrechold = 0.021 # 0.05
    wav.plotSelf(
        segments=wav.getSegments(wav.getEnergy(), energyThrechold)
      )
```

wav.py

```
import matplotlib.pyplot as plt
import numpy as np
import os
import wave

class WAV:
    def __init__(self, filePath):
        file = wave.open(filePath, 'rb')

        self.bitsPerSample = file.getsampwidth() * 8
```

```
        self.compressionName = file.getcompname()
        self.compressionType = file.getcomptype()
        self.filePath = filePath
        self.name = os.path.basename(filePath)
        self.numChannels = file.getnchannels()
        self.numSamples = file.getnframes()
        self.sampleRate = file.getframerate()
        self.samples = np.frombuffer(
            file.readframes(file.getnframes()),
            np.int16
        )

        self.frameDuration = 25 # in milliseconds

        file.close()

    def getAmplitudeMax(self) -> float:
        return self.samples.max()

    def getAmplitudeMin(self) -> float:
        return self.samples.min()

    def getBitrate(self) -> float:
        return self.sampleRate * self.numChannels * self.bitsPerSample

    def getChannelWidth(self) -> float:
        return 2 ** self.bitsPerSample

    def getChannelWidthMax(self) -> float:
        return self.getChannelWidth() / 2 - 1

    def getChannelWidthMin(self) -> float:
        return - self.getChannelWidth() / 2

    def getDuration(self) -> float:
        return self.numSamples / self.sampleRate

    def getEnergy(self):
        return self.getFrameFeatures(self.signalToEnergy)

    def getFileSizeComputed(self) -> float:
        return self.numSamples * self.numChannels * self.bitsPerSample / 8

    def getFrameFeatures(self, feature_extractor):
        window_size, hop_size = self.parseFrameDuration()

        samples = self.normalize(self.toMono())

        features = []
        for i in range(0, len(samples) - window_size + 1, hop_size):
            signal = samples[i:i + window_size]
            result = feature_extractor(signal)
            features.append(result)

        return self.normalize(np.array(features))

    def getSamplesForChannel(self, channelIndex):
        return self.samples[channelIndex::self.numChannels]

    def getSegments(self, data, threshold):
        window_size, hop_size = self.parseFrameDuration()

        segments = []
        segment_start = None
```

```
    step = (window_size - hop_size) / self.sampleRate
    for i, e in enumerate(data):
        if e > threshold:
            if segment_start is None:
                segment_start = i * step
        elif segment_start is not None:
            segment_end = i * step
            segments.append((segment_start, segment_end))
            segment_start = None

    return segments

def getZCR(self):
    return self.getFrameFeatures(self.signalToZCR)

def normalize(self, np_array):
    return np_array / np_array.max()

def parseFrameDuration(self):
    window_size = int(self.frameDuration * self.sampleRate / 1000)
    hop_size = window_size // 2

    return window_size, hop_size

def plot(self, data, label = "", title = ""):
    time_axis = np.linspace(0, self.getDuration(), len(data))

    plt.figure(figsize=(10, 4))
    plt.title(title)
    plt.plot(time_axis, data, linewidth=1)
    plt.xlabel('Time (s)')
    plt.ylabel(label)
    plt.grid()
    plt.show()

def plotSelf(self, cti: float = -1, segments = []):
    timeAxis = np.linspace(0, self.getDuration(), self.numSamples)

    plt.figure(figsize=(10, 4))

    for channelIndex in range(self.numChannels):
        plt.subplot(self.numChannels, 1, channelIndex + 1)

        plt.grid(True)
        plt.plot(timeAxis, self.getSamplesForChannel(channelIndex))
        plt.title(f'{self.name} (channel {channelIndex + 1})')
        plt.xlabel('Time (s)')
        plt.ylabel('Amplitude')
        # plt.ylim(self.getChannelWidthMin(), self.getChannelWidthMax())

        if cti > -1:
            plt.axvline(x=cti, color='r')

        for start, end in segments:
            plt.axvspan(start, end, color='red', alpha=0.5)

    plt.tight_layout()
    plt.show()

def setFrameDuration(self, frameDuration):
    self.frameDuration = frameDuration

def signalToEnergy(self, signal):
    return np.sum(np.square(signal))
```

```
def signalToZCR(self, signal):
    return np.sum(np.abs(np.diff(np.sign(signal)))) / (2 * len(signal))

def toMono(self):
    samples = self.samples
    if self.numChannels == 2:
        left = self.getSamplesForChannel(0)
        right = self.getSamplesForChannel(1)
        samples = (left + right) // 2

    return samples

def toString(self) -> str:
    return (
        f"bitRate: {self.getBitrate()} bits per second\n"
        f"bitsPerSample: {self.bitsPerSample} bits\n"
        f"compressionName: {self.compressionName}\n"
        f"compressionType: {self.compressionType}\n"
        f"duration: {self.getDuration()} seconds\n"
        f"filePath: {self.filePath}\n"
        f"fileSizeComputed: {self.getFileSizeComputed()} bytes\n"
        f"name: {self.name}\n"
        f"numChannels: {self.numChannels}\n"
        f"numSamples: {self.numSamples}\n"
        f"sampleRate: {self.sampleRate} Hz\n"
        f"samples: {self.samples}"
    )
```