

VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

INFORMACINIŲ SISTEMŲ INŽINERIJA

**Darbas Nr. 4**

**Trumpalaikio spektro analizė**

Garso signalų apdorojimas

Atliko: Paulius Zaranka

Vilnius

2022

1. DARBO TIKSLAS IR UŽDAVINIAI

**Darbo tikslas**. Garso signalų nuskaitymas, trumpalaikio spektro analizė, analizės rezultatų interpretavimas.

**Darbo uždaviniai.** Sukurti priemonę garso failams nuskaityti, nuskaitytiesiems signalams (ir jų atkarpoms) grafiškai atvaizduoti. Atlikti pasirinktosios signalo atkarpos spektro analizę. Įvertinti dažninę signalo sudėtį, išskirti esmines dažnines savybes.

1. DARBO REZULTATAI

Darbe analizuojami 3 skirtingi garso signalai: lietuviškai kalbančio vyro balso, operos vokalistės dainavimo ir miesto/gatvės šurmulio garso įrašai.

Pirmasis signalas – lietuviškai kalbančio vyro balsas. Žemiau pateiktos šio garso signalo laiko, 20 ms ilgio (nuo 2-os sekundės) atkarpos laiko ir spektro diagramos.

Chart, line chart

Description automatically generated

pav. lietuviškai kalbančio žmogaus balso laiko diagrama

Chart, line chart

Description automatically generated

pav. lietuviškai kalbančio žmogaus balso atkarpos laiko diagrama

Graphical user interface, text, application

Description automatically generated

pav. lietuviškai kalbančio žmogaus balso atkarpos spektro diagrama

Signalo atkarpoje vyrauja žemi (iki 500Hz) dažniai, kurie yra ryškiai stipriausi, ypatingai ties ~200Hz ir ~300Hz. Šiek tiek aptinkama ir vidutinių dažnių. Aukštų dažnių signale beveik nėra.

Antrasis signalas – operos vokalistės vokalas. Žemiau pateiktos šio garso signalo laiko, 20 ms ilgio (nuo 2-os sekundės) atkarpos laiko ir spektro diagramos.

Chart

Description automatically generated

pav. operos vokalo laiko diagrama

Chart, line chart

Description automatically generated

pav. operos vokalo atkarpos laiko diagrama

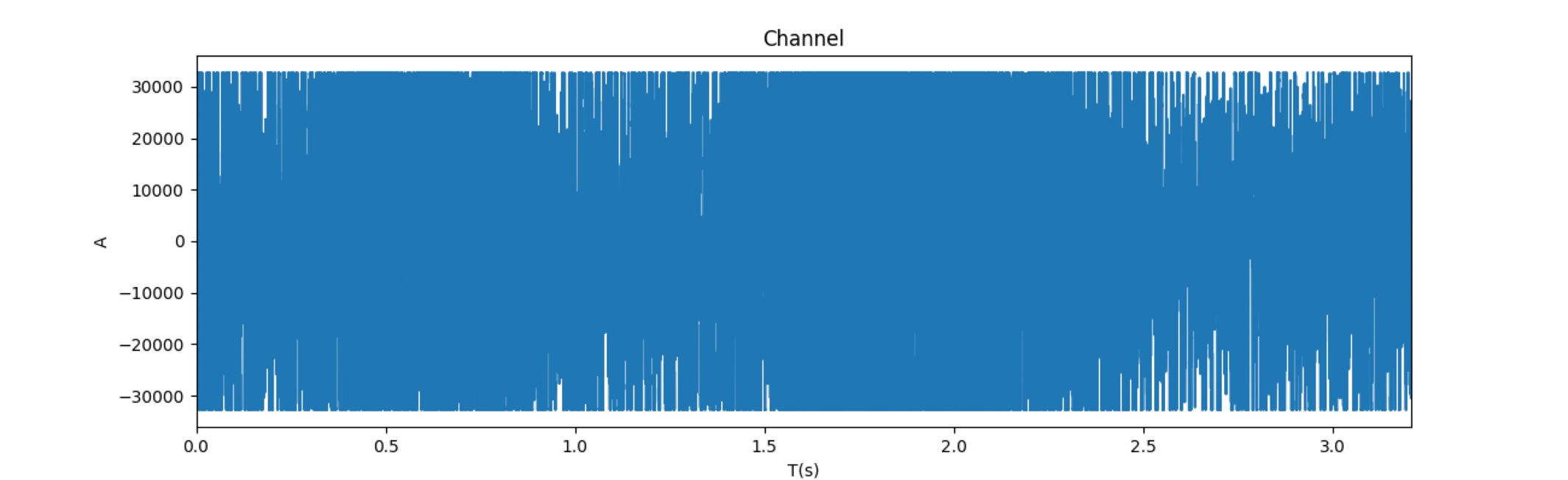
Graphical user interface, text, application

Description automatically generated

pav. operos vokalo atkarpos spektro diagrama

Signalo atkarpoje aptinkami ir žemi, ir vidutiniai, ir aukšti dažniai. Stipriausi jų – vidutiniai (šiame signale stipriausi nuo ~500Hz iki ~1000Hz). Galima aptikti ir dažnių, auktesnių nei 5000Hz, tačiau stipriausi dažniai varijuoja iki ~4000Hz.

Paskutinis analizuojamas signalas – miesto/gatvės šurmulys. Žemiau pateiktos šio garso signalo laiko, 20 ms ilgio (nuo 2-os sekundės) atkarpos laiko ir spektro diagramos.



pav. miesto/gatvės šurmulio laiko diagrama

A picture containing text, antenna, screenshot

Description automatically generated

pav. miesto/gatvės šurmulio atkarpos laiko diagrama

Graphical user interface, text, application

Description automatically generated

pav. miesto/gatvės šurmulio spektro diagrama

Signalo atkarpoje taip pat aptinkami ir žemi, ir vidutiniai, ir aukšti dažniai. Stipriausi jų – žemi, tačiau taip pat sąlyginai gana stiprūs yra ir vidutiniai, aukšti dažniai. Aptinkamas platus spektras dažnių, t.y. iki pat ~10kHz, bet nuo ~5000Hz matomas susilpnėjimas. Lyginant su kitais analizuojamais signalais, šis pasižymi didele garso galia, kai kur siekiančia beveik 8 mln. dB.

1. APIBENDRINIMAS IR IŠVADOS

Darbe analizuojant atskirų signalų trumpų atkarpų spektrus pasimatė jų sudedamosios dalys, t.y. dažniai ir jų garso galios. Lyginant signalus tarpusavyje išryškėjo skirtingų garsų charakteristikos. Darbo rezultatai patvirtino tai, ko būtų galima tikėtis dar iki analizės: vyriškas balsas yra žemesnis nei operos vokalistės, o gatvėje aptinkama labai įvairių ir stiprių garsų.

1. priedai

import math

import wave

from tkinter import filedialog

import matplotlib

import matplotlib.pyplot as plt

import numpy as np

import simpleaudio as sa

matplotlib.use('TkAgg') # to ignore warning message

def chooseFile():

return filedialog.askopenfilename(filetypes=(

("wav files", "\*.wav"), ("all files", "\*.\*")))

def readWav(path):

return wave.open(path, 'rb')

def playSound(file\_name):

wave\_obj = sa.WaveObject.from\_wave\_file(file\_name)

wave\_obj.play()

def getSegment(file, start\_time, end\_time):

framerate = file.getframerate()

n\_frames = file.getnframes()

frames = file.readframes(n\_frames)

frames\_array = np.frombuffer(frames, dtype=np.int16)

segment\_frames\_array = frames\_array[math.floor(

framerate \* start\_time):math.floor(framerate \* end\_time)]

return segment\_frames\_array

def visualizeChannels(frames\_array, times, l\_audio, n\_channels=1):

if n\_channels > 1:

visualizeStereoChannels(frames\_array, times, l\_audio, n\_channels)

return

visualizeMonoChannel(frames\_array, times, l\_audio)

def visualizeMonoChannel(values, times, length):

plt.figure(figsize=(13, 4))

plt.plot(times, values, zorder=1)

plt.title('Channel')

plt.ylabel('A')

plt.xlabel('T(s)')

plt.xlim(0, length)

plt.show()

def visualizeStereoChannels(values, times, length, n\_channels):

figure, plot\_channel = plt.subplots(n\_channels, figsize=(13, 6))

for i in range(n\_channels):

plot\_channel[i].set\_title(f"Channel #{i}")

plot\_channel[i].set\_ylabel('A')

plot\_channel[i].set\_xlabel('T(s)')

plot\_channel[i].plot(times, values[i::n\_channels], zorder=1)

plot\_channel[i].set\_xlim(0, length)

figure.tight\_layout()

plt.show()

def applyHamming(frames\_array):

hamming = np.hamming(len(frames\_array))

return np.multiply(frames\_array, hamming)

def plotOriginal(file\_name):

file = readWav(file\_name)

framerate = file.getframerate()

n\_frames = file.getnframes()

n\_channels = file.getnchannels()

l\_audio = n\_frames / framerate

b\_signal = file.readframes(n\_frames)

v\_signal = np.frombuffer(b\_signal, dtype=np.int16)

times = np.linspace(0, (n\_frames - 1) / framerate, num=n\_frames)

visualizeChannels(v\_signal, times, l\_audio, n\_channels)

file.close()

def plotSegment(file\_name):

start\_time = float(input('start time (s): '))

end\_time = float(input('duration (s): '))

file = readWav(file\_name)

segment = getSegment(file, start\_time, start\_time + end\_time)

framerate = file.getframerate()

n\_frames = len(segment)

l\_audio = n\_frames / framerate

times = np.linspace(0, (n\_frames - 1) / framerate, num=n\_frames)

visualizeChannels(segment, times, l\_audio)

def plotSpectrumFunction(file\_name):

start\_time = float(input('start time (s): '))

end\_time = float(input('duration (s): '))

file = readWav(file\_name)

frames\_array = getSegment(file, start\_time, start\_time + end\_time)

frames\_array\_windowed = applyHamming(frames\_array)

framerate = file.getframerate()

dft = np.fft.fft(frames\_array\_windowed)

dft\_abs = np.abs(dft)

if dft\_abs.size % 2 == 0:

cutted\_dtf = dft\_abs[:int((dft\_abs.size / 2) + 1)]

doubled\_dft = []

doubled\_dft.append(cutted\_dtf[0])

for i in range(1, len(cutted\_dtf) - 1):

doubled\_dft.append(cutted\_dtf[i] \* 2)

doubled\_dft.append(cutted\_dtf[len(cutted\_dtf) - 1])

else:

cutted\_dtf = dft\_abs[:int((dft\_abs.size + 1) / 2)]

doubled\_dft = []

doubled\_dft.append(cutted\_dtf[0])

for i in range(1, len(cutted\_dtf) - 1):

doubled\_dft.append(cutted\_dtf[i] \* 2)

dft\_frequences = np.fft.rfftfreq(

len(doubled\_dft) \* 2 - 1, 1 / framerate)

plt.figure(figsize=(15, 5))

plt.plot(dft\_frequences, doubled\_dft)

plt.title('Spectrum function')

plt.ylabel('Power (dB)')

plt.xlabel('Frequency (Hz)')

plt.show(block=False)

def interface(file\_name):

is\_running = True

while (is\_running):

print('1 - Plot original signal')

print('2 - Plot a segment of signal')

print('3 - Plot spectrum function of a segment')

print('4 - Play sound')

print('q - Quit program')

user\_input = input()

if (user\_input == '1'):

plotOriginal(file\_name)

if (user\_input == '2'):

plotSegment(file\_name)

if (user\_input == '3'):

plotSpectrumFunction(file\_name)

if (user\_input == '4'):

playSound(file\_name)

if (user\_input == 'q'):

is\_running = False

def main():

file\_name = chooseFile()

interface(file\_name)

main()