SKAITMENINIS SIGNALŲ APDOROJIMAS IR MAŠININIS MOKYMASIS 2022
Laboratorinis darbas nr. 1

DISKRETINIO LAIKO SISTEMŲ MODELIAVIMAS

Ž. Marma, E MEI-2 gr. Dėstytojas D. Sokas
KTU, Elektros ir elektronikos fakultetas

Įvadas

Laboratorinio darbo tikslas — išmokti modeliuoti diskretinio laiko sistemas ir tirti jų laikines bei dažnines charakteristikas, sprendžiant garsų apdorojimo problemą.

Laboratorinio darbo užduotis – sumodeliuoti gitaros akordo garsą bei garsus apdorojančius efektus ir ištirti laikines ir dažnines sumodeliuotų ir efektais apdorotų signalų charakteristikas. Laboratoriniam darbui realizuoti buvo naudojamas 10 (Dm) akordo numeris.

Natos signalo modeliavimas

Nustačius diskretizavimo dažnį lygų $f_d = 44100$ Hz buvo apskaičiuoti signalo vėlinimai kiekvienai natai naudojantis (1) formule. Gauti signalo vėlinimai pateikti lentelėje (1 lentelė.).

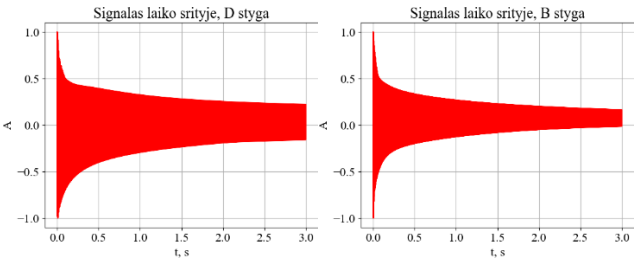
$$N = \frac{f_d}{f_s}. \quad (1)$$

1 lentelė. Natų vėlinimų vertės

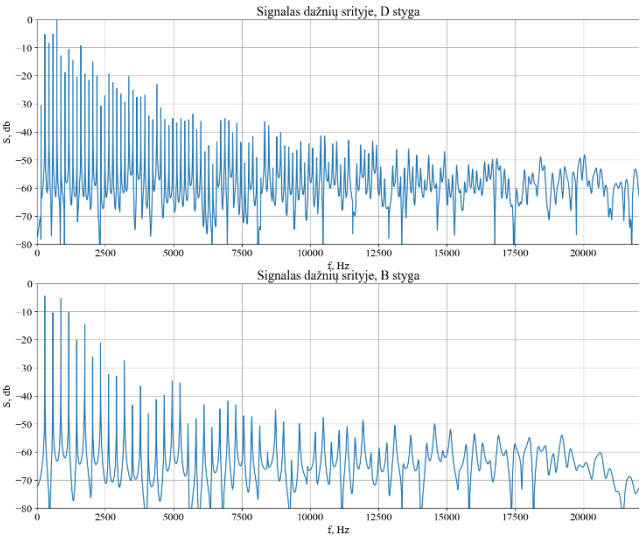
Styga	Stygos dažnis, Hz	Signalo vėlinimas (N), atskaita
A	110	401
D	147	300
G	220	200
B	294	150
e	349	126

Naudojantis struktūrine schema yra randami skaitmeninio filtro koeficientai b ir a:
b = [1], a = [1 *nuliuV* -0,5 -0,5], kur *nuliuV* yra nulių vektorius, kurio ilgis kiekvienai natai yra N.

Sumodeliavus natas, D ir B natų signalai yra grafiškai laiko ir dažnių srityse pateikti paveikslėliuose (analogiškai 1 pav. ir 2 pav.). Analizuojant D ir B stygų signalus laiko ašyje galima matyti, kad signalų forma yra ganėtinai panaši, tačiau iš grafikų (1 pav.) matyti, kad D stygos nusistovėjusi amplitudės dedamoji yra didesnė. Tuo galima įsitikinti ir klausantis sugeneruotus audio signalus, girdima, kad D stygos garsas garsesnis.



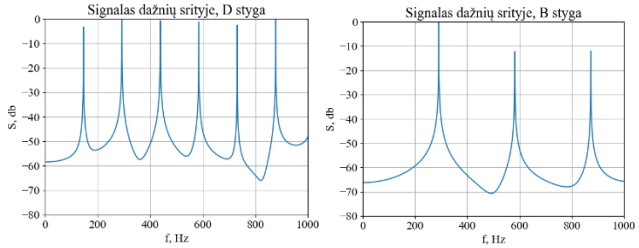
1 pav. D ir B stygos signalai laiko srityje.



2 pav. D ir B stygos signalų vaizdai dažnių srityje.

Analizuojant D ir B stygų spektrus galime pastebėti, kad D stygos spektras yra ženkliai tankesnis. Tai yra paaiškinama tuo, kad D stygos dažnis yra 147 Hz, o G – 294 Hz. Tai galime patvirtinti ir išanalizavus pirmąsias tris harmonikas. Analizuojant signalą nuo 0-500 Hz diapazone jau sutinkame tris D stygos harmonikas ir tik vieną B stygos harmoniką.

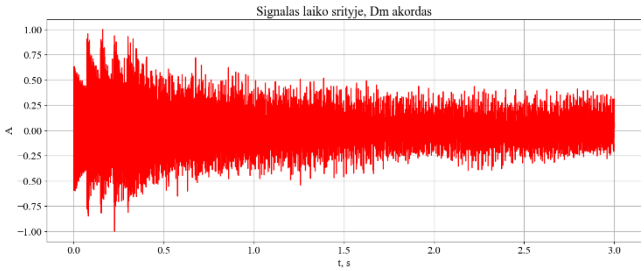
Išanalizavus D stygos signalą dažnių srityje pirmosios trys signalo harmonikos yra: 147 Hz, 294 Hz ir 440 Hz. Tuo tarpu B stygos pirmosios trys harmonikos yra: 293 Hz, 586 Hz, ir 879 Hz. Tai galima matyti grafiškai iš 3 pav. pateiktų priartintų signalų dažnių ašyje (harmonikos vertę laikome dažnį esantį ties pyku).



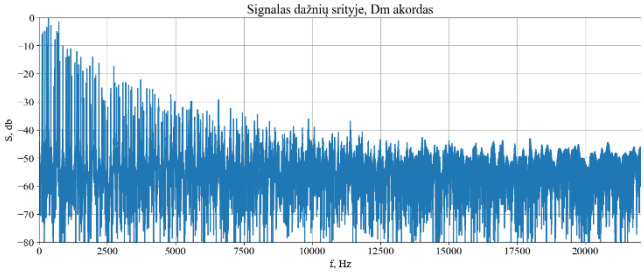
3 pav. D ir B stygų pirmosios harmonikos.

Akordo signalo modeliavimas

Atliekant akordo modeliavimą buvo pasirinktas 75ms vėlinimas tarp skirtingų natų ir atliktas natų sumavimas. Gauto signalo vaizdas pateiktas laiko (4 pav.) ir dažnių (5 pav.) srityse.

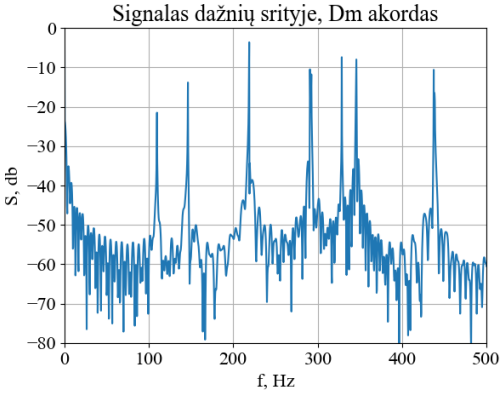


4 pav. Sumodeluotas Dm akordas laiko srityje.



5 pav. Sumodeluotas Dm akordas dažnių srityje.

Analizuojant sumodeliuoto akordo signalą laiko srityje (4 pav.) galima pastebėti, kiekvienos atskiros natos dedamąją (penki atskiri pykai per pirmąsias 500 ms). Analizuojant akordo signalą dažnių srityje (5 pav.) matome kad spektras yra ženkliai tankesnis palyginus su anksčiau analizuotais stygų vaizdais dažnių srityje (2 pav.). Tai atsitinka, nes signalas yra penkių skirtingo dažnio stygų suma.

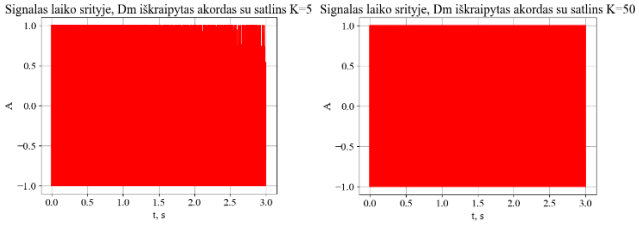


6 pav. Dm akordo pirmosios harmonikos.

Detaliau nagrinėjant signalo spektrą (6 pav.) galima matyti, kad tik pirmosios keturios akordo harmonijos sutapo su atskirų stygų virpėjimo dažniais: 110 Hz, 147 Hz, 220 Hz, 293 Hz, o penktosios harmonikos dažnis skyrėsi ir buvo lygus – 330 Hz. Taip atsitiko, nes žemiausios (A) stygos trečioji harmonika yra anksčiau dažnių srityje nei penktosios – e stygos dažnis (349 Hz).

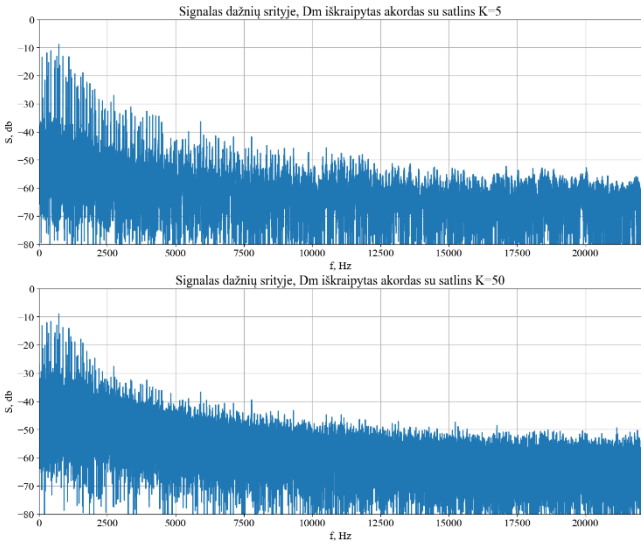
Iškraipymų efekto modeliavimas

Buvo pasirinktas koeficiento K vertė K=30, nes toks signalas atrodė priimtiniausias. Lyginant akordo skambesį, kai K yra lygi 5 su 50, esant K=50 galima girdėti daug pašalinio triukšmo garsas tampa nemalonus. Tuo tarpu kai K=5 girdimas ženkliai mažesnis efektas, tačiau akordas skamba vis tiek garsiau nei akordo signalas be iškraipymo efektų.



7 pav. Iškraipytas Dm akordas laiko srityje naudojant *satlins* (K=5 ir K =50).

Kaip galima matyti iš grafikų (7 pav.) signalų beveik visos reikšmės yra 1 arba -1. Vienintelis skirtumas yra naudojant stiprinimo koeficientą K =50 daugiau reikšmių buvo įsotintos, nes kai K = 5 galima matyti ties t ≈ 3s reikšmių kurių absoliutinę reikšmė liko mažesnė nei 1.

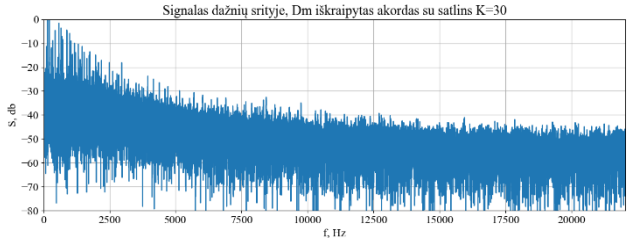


8 pav. Iškraipytas Dm akordas dažnių srityje naudojant *satlins* (K=5 ir K =50).

Palyginus iškraipytus signalus dažnių srityje (8 pav.) galima matyti, jog esant K = 50 prie žemų dažnių signalas yra stipresnis (mažiau slopinamas).

Lyginant Dm akordo signalą prieš apdorojimą iškraipymų efektu (4 pav.) ir po apdorojimo/iškraipymo (9 pav.) galima matyti ženklius skirtumus: iškraipymas neleidžia matyti atskirų stygų sukuriamus amplitudės šuolius. Analizuojant akordą dažnių srityje po apdorojimo (10 pav.) galima pastebėti padidėjusį harmonikų skaičių lyginant su neapdorotu signalu (5 pav.).

9 pav. Dm akordo signalas po iškraipymų laiko srityje naudojant *satlins* (K=30).



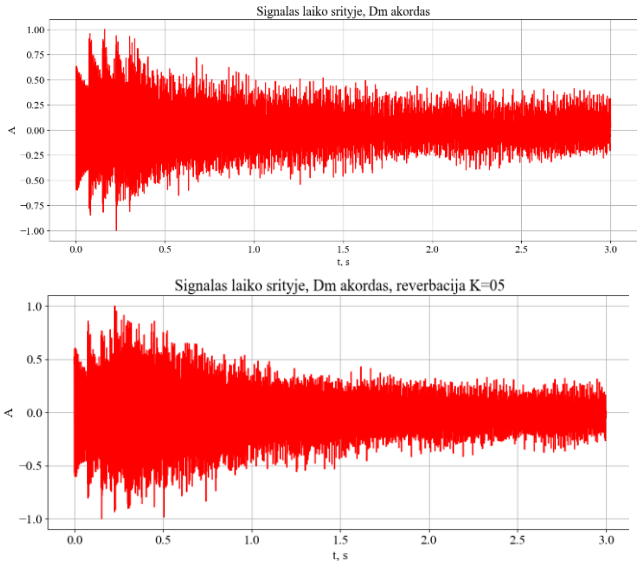
10 pav. Dm akordo signalas po iškraipymų dažnių srityje naudojant *satlins* (K=30).

Reverberacijos efekto modeliavimas

Laboratoriniame darbe taip pat buvo modeliuojamas garso reverberacijos efektas, kuris imituoja daugkartinius garso atspindžius nuo atspindinčių patalpos paviršių. Norint realizuoti šio tipo efektą buvo suprojektuotas skaitmeninis filtras su šiomis koeficientų b ir a reikšmėmis:
b = [1], a = [1 *nuliuV -K*], kur *nuliuV* yra nulių vektorius, kurio ilgis lygus signalo vėlinimui (pasirinkta 200ms vertė), o *K* slopinimo koeficientas (pasirinktas 0,5).

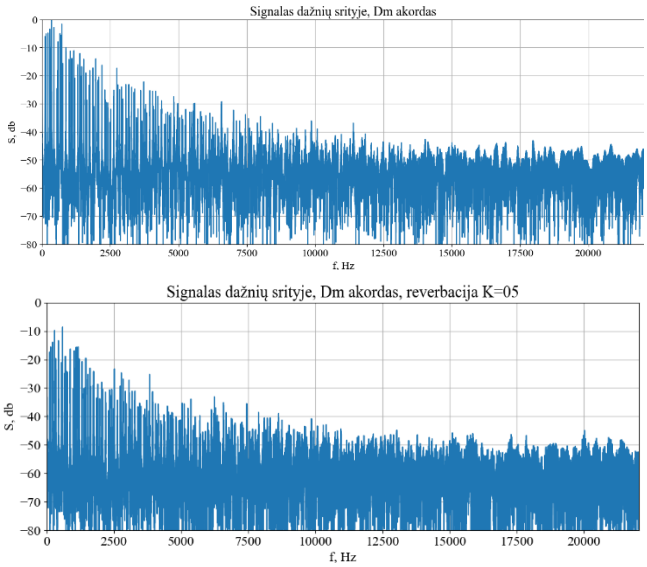
Eksperimentiškai keičiant slopinimo koeficientą nuo 0 iki 1 galima pastebėti, kad reverberacijos efekto stiprumas yra tiesiogiai proporcingas šiai vertei. Esant K=0 negirdimas reverberacijos efektas, o kai K = 1 girdimas itin stiprus ir pasikartojantis aido efektas (tai atsitinka dėl teigiamojo grįžtamojo ryšio). Tačiau maloniausias efektas pasiekiamas esant 0,5 ar 0,4 koeficiento reikšmei.

Analizuojant reverberacijos efektą akordo signalui laiko srityje (11 pav.), galima matyti amplitudės padidėjimą po vėlinimo (pasirinkta 200ms vertė) lyginant su originaliu signalu. Taip yra pasiekiamas aido efektas garsui.



11 pav. Orginalus Dm akordo signalas ir Dm akordo signalas pritaikius reverberacijos efektą laiko srityje.

Apdorojus signalą reverberacijos efektu matomas signalo patankėjimas dažnių srityje (12 pav.). Šis harmonikų padaugėjimas ir yra vis pasikartojantys stygų garsai.



12 pav. Orginalus Dm akordo signalas ir Dm akordo signalas pritaikius reverberacijos efektą dažnių srityje.

Papildoma užduotis

Laboratoriniame darbe buvo pasirinktas a variantas ir įgyvendinti pažangesni iškraipymo efektai – *overdrive* ir *fuzz*.

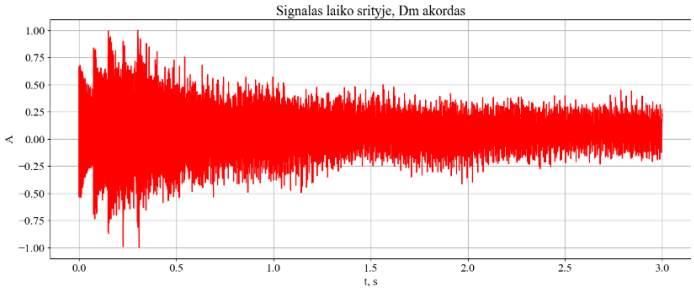
Overdrive iškraipymų efekto modeliavimas

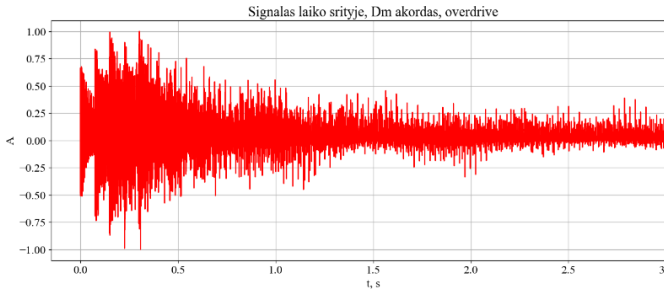
Įsigilinus į *overdrive* iškraipymo lygtį akivaizdu, kad signalo, reikšmės kurių absoliutinė vertė yra mažesnė nei $\frac{2}{3}$ bus slopinamos. Kadangi koeficientai *overdrive* funkcijoje neviršija vieneto (2 ir 3 formulės)

$$k = 2|x|, \qquad kai\ 0 \le |x| < \frac{1}{3}, \tag{2}$$

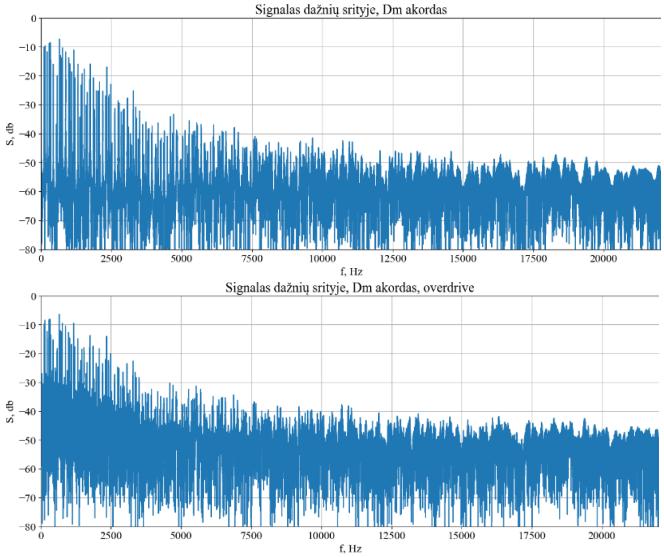
$$k = \frac{3 - (2 - 3|x|)^2}{3}, \qquad kai\ \frac{1}{3} \le |x| < \frac{2}{3}. \tag{3}$$

Analizuojant *overdrive* iškraipymo efektą laiko srityje (13 pav.) lengvai galima pastebėti, kad silpnas garsas esantis signale po 1,2s yra labiausiai slopinamas. Galima teigti, kad šis efektas veikia tarsi filtras silpninantis mažos amplitudės signalo vertes. Išnagrinėjus *overdrive* efektu iškraipytą signalą dažnių srityje (14 pav.), galima matyti, kad signalai esantys tarp 0-5000Hz yra mažiau slopinami palyginus su originaliu akordu.





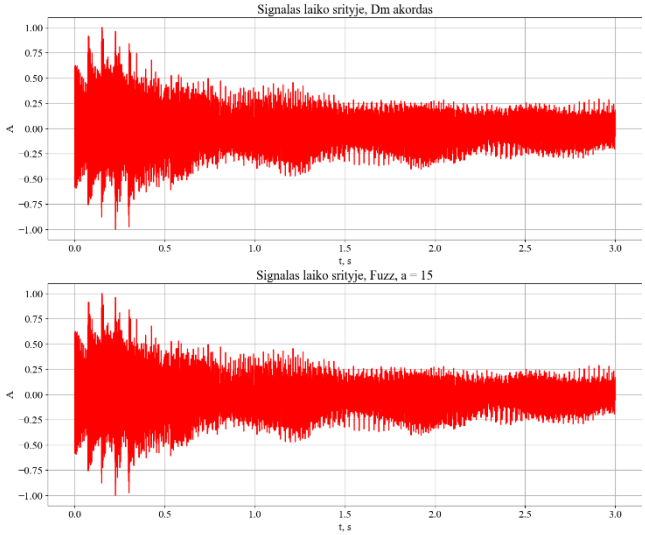
13 pav. Orginalus Dm akordo signalas ir Dm akordo signalas pritaikius *overdrive* efektą laiko srityje.



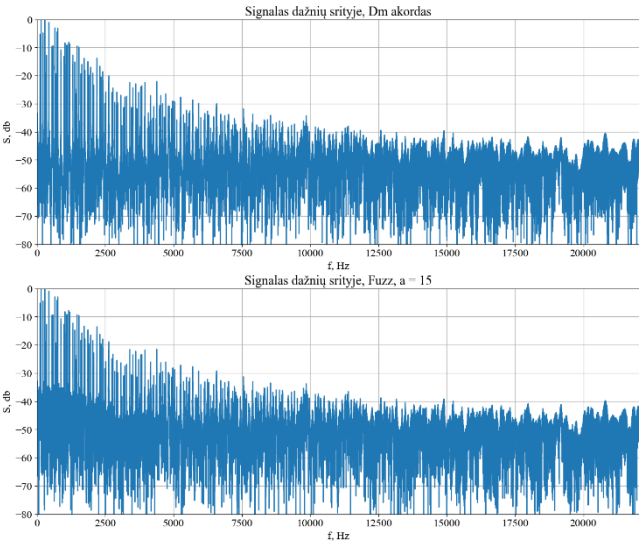
14 pav. Orginalus Dm akordo signalas ir Dm akordo signalas pritaikius *overdrive* efektą dažnių srityje.

Fuzz iškraipymų efekto modeliavimas

Įgyvendinant fuzz efektą buvo pasirinktas stiprinimo koeficientas a = 15. Šis iškraipymo efektas labiausiai slopina signalo vertes su maža amplitude, o įėjimus kurių vertė yra artima vienetui slopina mažiausiai. Laiko srityje (15 pav.) matomas tik nedidelis amplitudės sumažėjimas, signalo vertėms kurių amplitudės ir taip mažos. Dažnių srityje (16 pav.) galima matyti tolygesnį slopinimo išsidėstymą ties žemais dažniais (0-3000 Hz).



15 pav. Orginalus Dm akordo signalas ir Dm akordo signalas pritaikius *fuzz* efektą laiko srityje.



16 pav. Orginalus Dm akordo signalas ir Dm akordo signalas pritaikius *fuzz* efektą dažnių srityje.

Remiantis perdavimo funkcijomis garso efektas kuriam buvo panaudota *satlins* funkcija visas signalo vertes daugina iš K koeficiento ir tada normuoja tarp -1 ir 1, todėl esant K daugiau nei 5 signalo amplitudė įgauna ribines vertes. *Fuzz* ir *overdirve* metodai taip neiškreipia amplitudės kaip *satlins*, tačiau abu metodai silpnina mažos amplitudės vertes ($|x| < 0{,}8$). Verta paminėti, kad *overdrive* labiau silpnina itin mažos amplitudės vertes ($0 < |x| < 0{,}5$).

Rezultatai

Laboratoriniame darbe buvo sumodeliuotos natas, užduotas akordas taip įgyvendintas natų vėlimo ir sudėjimo algoritmas. Detaliai išnagrinėti garsų apdorojimo efektai, naudojantis *satlins* pagrindu buvo gautas būdingas „sunkus“ gitaros skambesys. Palyginti akordo skambesiai, kai K yra lygi 5 su 50, esant K=50 buvo galima girdėti daug pašalinio triukšmo. Tuo tarpu kai K=5 girdimas ženkliai mažesnis efektas, tačiau akordo skambesys yra malonus ir su efektu.

Buvo sumodeliuotas reverberacijos efektas. Eksperimentiškai keičiant slopinimo koeficientą nuo 0 iki 1 buvo pastebėta, kad reverberacijos efekto stiprumas yra tiesiogiai proporcingas šiai vertei. Esant K=0 negirdimas reverberacijos efektas, o kai K = 1 girdimas itin stiprus ir pasikartojantis aido efektas, tam darė įtaką teigiamas grįžtamasis ryšis.

Įgyvendinta papildoma užduotis ir sumodeliuoti *fuzz* ir *overdrive* iškraipymo efektai. Išanalizuoti ir pateikti skirtumai tarp netiesinių iškraipymų naudojant *satlins* funkciją, *overdrive* efektą ir *fuzz* efektą.

Išvados

Laboratoriniame darbe buvo atliktas gitaros akordo garso bei garsus apdorojančių efektų modeliavimas. Ištirtos laikines ir dažnines sumodeliuotų ir efektais apdorotų signalų charakteristikos.

Priedai

Python programos kodas:

```
# student number = 10
# accord = Dm
# f1 = 0 # is not used

import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy.io.wavfile import write
from sklearn import preprocessing

class MusicLab:
    def __init__(self, debug = False):
        self.debug = debug

        f2 = 110
        f3 = 147
        f4 = 220
        f5 = 294
        f6 = 349
        self.notes = [f2, f3, f4, f5, f6]
        self.notesNames = ["A styga", "D styga", "G styga", "B styga", "e styga"]
        self.STRING_COUNT = 5
        self.saveDir = "out/"

        self.samplingRate = 44100
        self.t_s = 3
        self.N = np.empty(self.STRING_COUNT, dtype=int)

    def countDelays_N(self):
        for i, note in enumerate(self.notes):
            self.N[i] = round(self.samplingRate / note) # 1 Task

    def generateInput_X(self): # 2 Task
        x_final = []
        x_random = []

        for i in range(0, self.STRING_COUNT):
            x_random = np.random.uniform(0, 1, self.N[i])
            K_zeroCount = self.samplingRate*self.t_s - self.N[i]
            if self.debug:
                print(K_zeroCount)

            x_zeros = []
            x_zeros = np.zeros(K_zeroCount)

            x_final.append(np.concatenate([x_random, x_zeros]))
            if self.debug:
                print(x_final[i])
        return x_final

    def generateSound_Y(self, signal_x):
        y_final = []
        for i in range(0, self.STRING_COUNT):
            b = [1]
            a = np.concatenate([[1], np.zeros(self.N[i]), [-0.5, -0.5]])

            audioData = signal.lfilter(b, a, signal_x[i])
            audioScaled = preprocessing.minmax_scale(audioData, feature_range=(-1,1))
            y_final.append(audioScaled); # 3 Task

        if self.debug:
            print(np.shape(signal_x))
        # static assert ar len == (fd*ts=44100*3=132300)
        return y_final

    # 4. Listen to notes:
    def saveNoteAsWav(self, noteData, filename):
        write(filename=filename, rate=self.samplingRate, data=noteData.astype(np.float32))

    def drawSignal(self, signal_y, title, show=False):
        tn = np.linspace(0, self.t_s, num=len(signal_y))
        plt.figure
        plt.plot(tn, signal_y, 'r-')
        plt.title('Signalas laiko srityje, ' + title)
        plt.xlabel('t, s')
        plt.ylabel('A')
        plt.grid(True)
        plt.savefig(self.saveDir + "amp_" + title,  bbox_inches = 'tight',
                    pad_inches = 0)
        if show:
            plt.show()
        plt.close()

    # 5 Get FFT and draw spectrum
    def drawSpectrum(self, signal_y, title, show = False):
        nfft = len(signal_y)
        yf = np.fft.fft(signal_y)

        spectrum = np.abs(yf) / nfft
        spectrum_db = 20 * np.log10(spectrum/np.max(spectrum))

        k = list(range(0, nfft))
        f_Hz = [i * (self.samplingRate/nfft) for i in k]

        ax = plt.axes()
        ax.plot(f_Hz, spectrum_db)
        ax.set_xlim(0, self.samplingRate/2)
        ax.set_ylim(-80, 0)
        plt.title('Signalas dažnių srityje, ' + title)
        plt.xlabel('f, Hz')
        plt.ylabel('S, db')
        plt.grid(True)
        plt.savefig(self.saveDir + "spec_" + title,  bbox_inches = 'tight',
                    pad_inches = 0)
        if show:
            plt.show()
        plt.close()

    # 3.1.2 Simulate accord
    def generateAccord(self, allNotes):
        delay_ms = 75
        second_ms = 1000
```

```
        n_delay = (delay_ms * self.samplingRate / second_ms)
        # delay each note for some time (delay time: first t, second t*2, t*3, .... t*N)

        notesWithDelay=[]
        for i, note in enumerate(allNotes):
            if i == 0: # do not delay first note
                notesWithDelay.append(np.concatenate([note]))
            else:
                count = int(i * n_delay)
                temp_a = note[:-count]
                delayZeros = np.zeros(count).astype(np.float64)

notesWithDelay.append(np.concatenate([delayZeros, temp_a]))
        accord = np.zeros(len(allNotes[0])).astype(np.float64)
        for note in notesWithDelay:
            accord = accord + note
        accord = preprocessing.minmax_scale(accord, feature_range=(-1,1))
        return accord

    # 3.2.1 Distortion using satlins
    def nonLinearDistortion(self, signal):
        def satlins(n):
            if (n <= -1):
                return -1
            if (-1 <= n <= 1):
                return n
            if (1 <= n):
                return 1
        sig_after = [satlins(i) for i in signal]
        return sig_after

    """ 3.2.2 Modeling the reverberation effect
        # filter coefficients
        # b = [1]
        # a = [1; 0...0(N); -(K)]
    """
    def addReverb(self, signalIn, N_ms, K_coef):
        second_ms = 1000
        n_delay =  (N_ms * self.samplingRate / second_ms)
        b = [1]
        a = np.concatenate([[1], np.zeros(int(n_delay)), [-K_coef]])
        reverbedSignal = signal.lfilter(b, a, signalIn)
        reverbedSignal = preprocessing.minmax_scale(reverbedSignal, feature_range=(-1,1))
        return reverbedSignal

    def analyzeDistortion(self, accord_in, k):
        accordSignal_mod = np.multiply(accord_in, k)
        distortedAccord = self.nonLinearDistortion(accordSignal_mod)
        distortedAccord = np.multiply(distortedAccord, 1) # some how it needs to change type
        self.drawSignal(distortedAccord, f"Dm iškraipytas akordas su satlins K={k}")
        self.drawSpectrum(distortedAccord, f"Dm iškraipytas akordas su satlins K={k}")
        self.saveNoteAsWav(distortedAccord, f"DistAccord{k}.wav")


""" 4. a)
Overdrive is an effect where the amplitude of the input signal undergoes a non-linear amplification. The threshold determines how much of
the signal undergoes the nonlinear amplification curve (lower threshold implies more of the signal is captured in the curve).
"""
def overdrive(signal):
    def overLambda(x):
        # Remove a result of floating-point approximation for python floats
        if x > 1 and x < 1.000000000000004:
            x = 1
        elif x < -1 and x > -1.000000000000004:
            x = -1

        if(np.abs(x) > 1):
            print(x)


        if (0 <= np.abs(x) and np.abs(x) < (1/3)):
            return x * 2 * np.abs(x)
        elif ((1/3) <= np.abs(x) and np.abs(x) < (2/3)):
            return x * (3-(2-(3*np.abs(x)))**2) / 3
        elif ((2/3) <= np.abs(x) and np.abs(x) <= 1):
            return x
        else:
            print("Incorrect value!!! (out of range -1:1)")
            print(x)

    sig_after = [overLambda(i) for i in signal if overLambda(i) is not None]
    sig_after = np.multiply(sig_after, 1) # That multiplication changes data type to list
    return sig_after

def applyFuzz(signal, a = 50):
    fuzz = lambda x: x * (1 - np.exp(-a * np.abs(x)))
    sig_after = fuzz(signal)
    return sig_after

############### main

plt.rcParams.update({'font.family': "Times New Roman"})
plt.rcParams.update({'font.size': 16})
plt.rcParams.update({'figure.figsize': (16, 6)}) # horizontally longer figure

musicObj = MusicLab()
musicObj.countDelays_N()
print(f"Signal delays {musicObj.N}")

signals_X = musicObj.generateInput_X()
sounds_Y = musicObj.generateSound_Y(signals_X)

for i, y_sig in enumerate(sounds_Y):
    musicObj.drawSignal(y_sig, musicObj.notesNames[i])
```

```
    musicObj.drawSpectrum(y_sig,
musicObj.notesNames[i])
    musicObj.saveNoteAsWav(y_sig, f"Note{i}.wav")

accordSignal = musicObj.generateAccord(sounds_Y)
musicObj.drawSignal(accordSignal, "Dm akordas")
musicObj.drawSpectrum(accordSignal, "Dm akordas")
musicObj.saveNoteAsWav(accordSignal, "accord.wav")

# Analyze how the sound of the chord and its temporal and
frequency characteristics change when K = 5 and K = 50.
musicObj.analyzeDistortion(accordSignal, k=5)
musicObj.analyzeDistortion(accordSignal, k=50)

# 3.2.2 task
N_ms = 200
K_reverb = 0.5

accordSignal = musicObj.generateAccord(sounds_Y)
accordSignal_s = musicObj.addReverb(accordSignal,
N_ms, K_reverb)

musicObj.drawSignal(accordSignal_s, f"Dm akordas,
reverbacija K=05") # ahhh Python cannot parse that dot in
number
musicObj.drawSpectrum(accordSignal_s, f"Dm akordas,
reverbacija K=05")
musicObj.saveNoteAsWav(accordSignal_s,
f"Reverb.wav")


# 4 extra task
accordSignal_o = musicObj.generateAccord(sounds_Y)
musicObj.drawSignal(accordSignal_o, "Dm akordas")
musicObj.drawSpectrum(accordSignal_o, "Dm akordas")

accordSignal_over = overdrive(accordSignal_o)
musicObj.drawSignal(accordSignal_over, "Dm akordas,
overdrive") # TODO: check if thats really zero
musicObj.drawSpectrum(accordSignal_over, "Dm akordas,
overdrive")
musicObj.saveNoteAsWav(accordSignal_over,
f"Overdrive.wav")


accordSignal_f = musicObj.generateAccord(sounds_Y)
a = 15
accordSignal_fuzz = applyFuzz(accordSignal_f, a)
musicObj.drawSignal(accordSignal_fuzz, f"Fuzz, a = {a}")
musicObj.drawSpectrum(accordSignal_fuzz, f"Fuzz, a =
{a}")
musicObj.saveNoteAsWav(accordSignal_fuzz, f"Fuzz
a={a}.wav")
```