SKAITMENINIŲ SIGNALŲ APDOROJIMAS 2020
Laboratorinis darbas nr. 1

Diskretinių laiko sistemų modeliavimas

Lukas Stasytis, E MEI-0 gr. Dėstytojas prof. V. Marozas
Kauno technologijos universitetas, Elektros ir elektronikos fakultetas

Įvadas

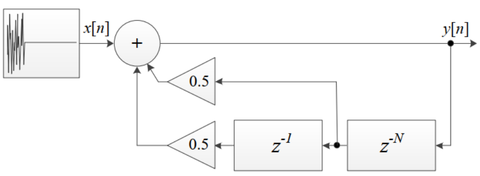
Šio laboratorinio darbo tikslas yra išmokti modeliuoti diskretinio laiko sistemas ir tirti jų laikines bei dažnines charakteristikas. Šiam tikslui įgyvendinti, modeliuosime gitaros stygų akordą ir pritaikysime iškraipymo bei reverberacijos efektus. Taip pat pritaikysime amplitudinę ir žiedinę moduliacijas.

Tolesnėse skiltyse bus aptariami individualūs signalų apdorojimo metodai su rezultatais pasekoje kiekvienos skilties. Pradėsime nuo vienos natos modeliavimo, tada pareisime prie pilno akordo ir galiausiai pritaikysime įvairius moduliavimo metodus sumodeliuoto akordo signalui transformuoti.

Vienos natos modeliavimas

Karplus ir Strong algoritmas

Gitaros skambesiui išgauti naudosimės Karplus ir Strong styginių instrumentų garsų sintezės algoritmu [1]. Gitaros modeliavimui bus pritaikomas neribotos impulsinės reakcijos (NIR) filtras. (pav 1). Iš struktūrinės schemos galime pastebėti, kad mums reikės vėlinimo koeficiento N bei filtro koeficientų a ir b.



1 pav. Gitaros natos modeliavimo schema.

NIR filtro vėlinimo radimas

Visų pirma, turime surasti natos vėlinimo koeficientą, kuris reikalingas NIR filtrui. Šis koeficientas, mums nurodys per kiek atskaitų turime žiūrėti atgal savo jau sugeneruotas išvestis, generuojant naujas išvestis. Pavyzdžiui, jeigu turėtume $N = 3$ ir naudotume elementarų filtrą, tokį kaip formulėje 1, mūsų sekanti atskaitos išėjimo reikšmė būtų lygi trimis reikšmėmis seniau sugeneruotai reikšmei. Generavimą pradėsime po pradinės atsitiktinio trukšmo aibės, dėl to už aibės ribų neišeisime.

$$y[n] = y[n - N] \tag{1}$$

NIR filtro vėlinimą galime surasti garso signalo diskretizavimo dažnį (f_d) padalinus iš stygos virpėjimo dažnio (f_s): lygtis 2

$$N = \frac{f_d}{f_s} \tag{2}$$

Mūsų atveju, diskretizavimo dažnis f_d = 44100 Hz. Pirmoji nata kurios signalą generuosime turi virpėjimo dažnį f_s = 165 Hz. Svarbu paminėti, kad vėlinimo reikšmė turi būti sveikas skaičius, nes, tai reikšmė nurodanti kelintą narį naudosime. Dėl to suapvalinsime gautą rezultatą.

3 pateikiamas D natos vėlinimo apskaičiavimas:

$$N_D = round(\frac{44100}{165}) = 267 \tag{3}$$

A ir B koeficientų radimas

Ieškant A ir B koeficientų, pasinaudosime gitaros natos modeliavimo schemai atitinkančia skirtumine lygtimi 4

Signalo įėjimo atskaitos žymimos x simboliu, o įšėjimo - y.

$$y[n] = x[n] + \frac{y[n - N] + y[n - N - 1]}{2} \tag{4}$$

Lygtį pasikeičiame į Z ašį ir išsikeliame įėjimo signalo atskaitas į lygties viršų, o įšėjimo atskaitas - apačią, kaip matoma 5, 6 bei 7 lygtyse. Verta pastebėti, kad apatinės lygties reikšmės keičia ženklą ir pridedamas vienetas.

$$y[z] = x(z)^0 + 0.5y(z)^{-N} + 0.5y(z)^{-N-1} \tag{5}$$

$$H[z] = \frac{Y[z]}{X[z]} = \frac{x(Z)^0}{0.5y(Z)^{-N} + 0.5y(Z)^{-N-1}} \tag{6}$$

$$H[z] = \frac{1}{1 - 0.5z^{-N} - 0.5z^{-N-1}} \tag{7}$$

a koeficientas bus lygties viršutinis narys, ženklo nekeičiant.

b koeficientais bus lygties apatiniai nariai. Visi nenaudojami nariai lygus nuliui. Šiuo atveju, jų kiekis bus lygus N-3.

$$b = [1, 0, ..., 0, -0.5, -0.5] \tag{8}$$
$$a = [1] \tag{9}$$

Signalo generavimas

Signalą generuosime $t_s = 3$ s trukmės. Signalo pirmos N reikšmių turi būti atsitiktinis triukšmas intervale [0,1]. Likusios reikmės iš pradžių turėtų būti lygios 0. Šių reikšmių kiekį galime sužinoti iš bendro signalo diskretinių taškų kieko atėmus triukšmo kiekį. Diskretinių reikšmių kiekis yra lygus tiesiog diskretizavimo dažnio bei signalo trukmės sekundėmis sandaugai:

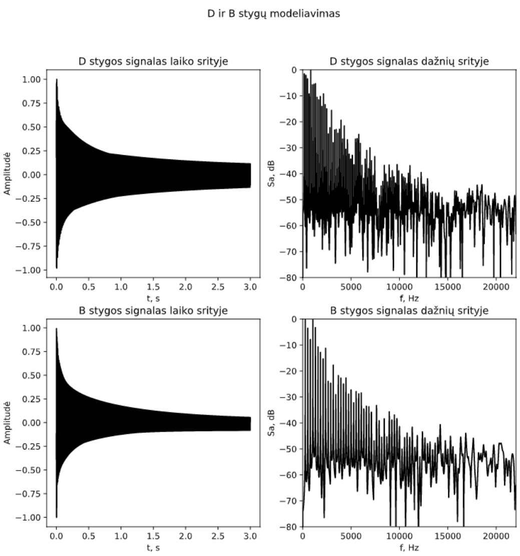
$$K_D = f_d \cdot t_s - N_D \qquad (10)$$

$$K_D = 44100 \cdot 3 - 267 = 132033 \qquad (11)$$

Galiausiai, gautą signalą normalizuojame į intervalą [-1,1]. Tai atliekame iš signalo atimdami jo vidurkį ir gautą signalą padalindami iš gauto signalo modulio maksimumo.

Visoms operacijoms naudojame python programavimo kalbos numpy,scipy bei pyplot paketus[2],[3],[4]. NIR filtrui panaudojame scipy funkciją $lfilter$ [5] kuri atitinka matlab funkciją $filter$.

Pav 2. pavaizduotos sumodeliuotos D ir B stygos. Informacija pateikiama laiko bei dažnių srityse. Matome laike slopstantį signalą su didžiaja dalimi diskretinių reikšmių pasiskirčiusių apie -50db. Taigi, signalas iš pradžių turėjo ženklų aukštos amplitudės garsą ir tada pradėjo slopti.



2 pav. Gitaros natų D ir B laiko ir dažnių grafikai

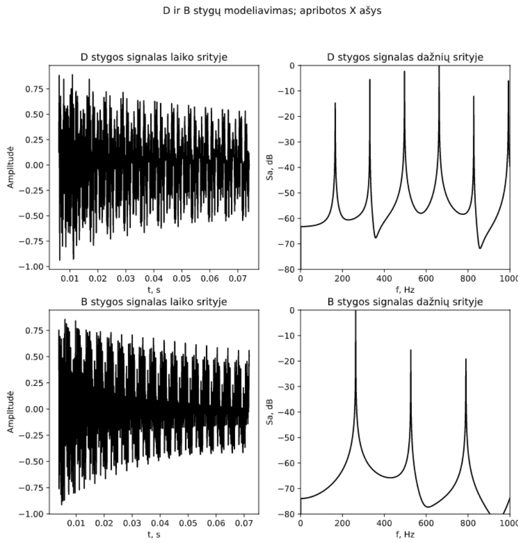
Norėdami geriau paanalizuoti signalus, apribosime laiko srities atvaizduotas atskaitas į pirmų 0.07 sekundžių po atsitiktinio triukšmo, o dažnių srities - pirmus 1000 Hz. Rezultatai pavaizduoti Pav 3.

Stebint dažnių sritis, iš karto galime pastebėti, kaip susidariusios harmonikos turi periodą atitinkantį simuliuojamų stygų dažniams. D stygos dažnis 165 Hz bei B stygos dažnis 262 Hz sudaro proporcingai beveik dvigubai ilgesnius stygų dažnių periodus.

Tuo metu laiko srityje matosi tie patys periodai amplitudėje, kurie daug tankesni D stygos atveju.

Akordo generavimas

Akordui generuoti naudosime penkias stygas: A,D,G,B,e. Stygų dažniai pateikiami lentelėje nr 1.



3 pav. Gitaros natų D ir B laiko ir dažnių grafikai apribojus X ašis.

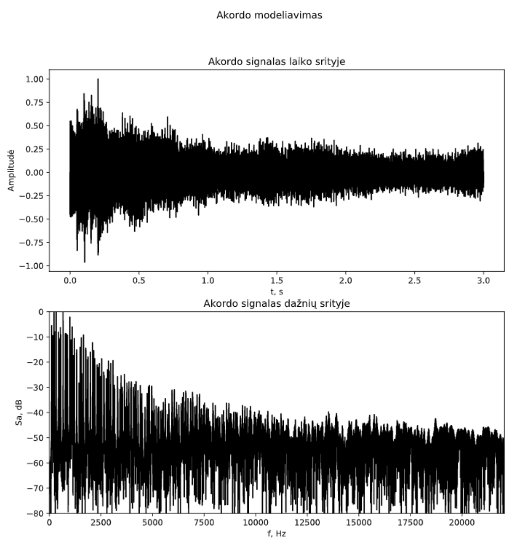
1 lentelė. Generuojamą akordą sudarančių stygų dažnių lentelė

	A	D	G	B	e
Hz	110	165	220	262	330

Akordui sugeneruoti, mes generuojame kiekvieną individualią stygą, ją normuojame, suvėliname per 50ms ir tada susumuojame signalus. Vėlinimas realizuojamas pastumiant kiekvienos stygos signalą į dešinę per 50ms atitinkančius diskretizavimo taškus pateiktam signalo diskretizavimo dažniui, tada likusias laisvas vietas kairėje užpildant nulinėmis reikšmėmis.

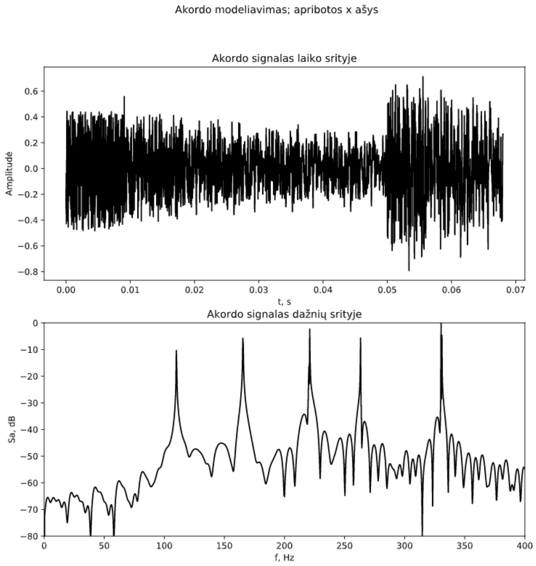
Reikšmių sumavimas atliekamas elementariai susumuojant individualių stygų amplitudes ties kiekvienu diskretinių tašku.

Pav 4. pavaizduotas sumodeliuotas stygų akordas. Informacija pateikiama laiko bei dažnių srityse. Matome daug triukšmingesnius signalus, negu individualių stygų atveju. Kiekviena styga įvedė savo papildomo triukšmo į signalą.



4 pav. Gitaros akordo sumodeliuoto signalo laiko ir dažnių grafikai

Pav 5. pateikiamas apribotos X ašies vaizdas. Galime pastebėti pirmas penkias harmonikas dažnių srityje bei antros, suvėlintos, stygos pradžia praėjus 50ms po pirmosios generavimo pradžios. Tarpai tarp harmonikų sąlyginai sutampa su pačių natų dažniais.



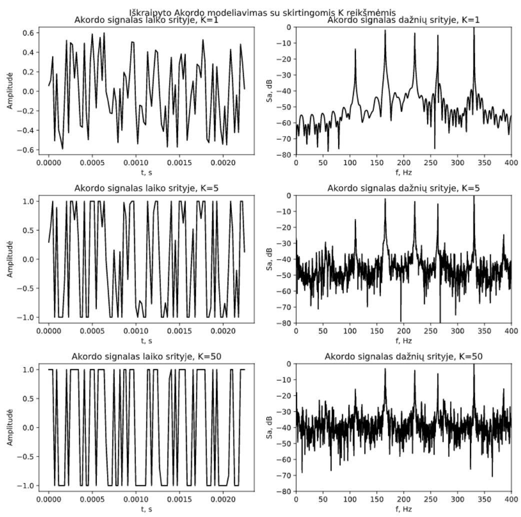
5 pav. Gitaros akordo sumodeliuoto signalo laiko ir dažnių grafikai apribojus X ašis.

Papildomų efektų modeliavimas

Iškraipymų efektas

Iškraipymo efektui išgauti sustiprinsime ankstesniu akordo generavimo metodu išgautą signalą K kartų ir tada apribosime jo amplitudę [-1,1] ribose. Eksperimentui panaudosime tris K reikšmes: [1,5,50].

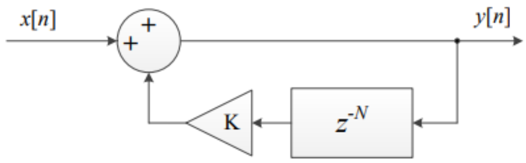
6 pav. pavaizduoti šio stiprinimo rezultatai žiūrint tik į pirmus kelis šimtus reikšmių. Matomas ryškus amplitudžių 'suaštrėjimas', daugeliui tolygių perėjimų iš neigiamų reikšmių į teigiamas pavirtus į staigius perėjimus. Dažnių srityje signalas taip pat prarado daug savo tolygumo perėjimuose. Susidarė savotiškas signalo triukšmas. $K = 5$ koeficiento signalas dar yra pakenčiamas ir skamba kaip roko muzikos, tačiau keliant reikšmę link $K = 50$ garsas tampa tiesiog aukšto dažnio triukšmu.



6 pav. Gitaros akordo sumodeliuoto signalo laiko ir dažnių grafikai su iškraipymo efektais.

Reverberacijos efektas

Reverberacijos efektui išgauti naudosime skaitmeninį neribotos impulsinės reakcijos filtrą. Struktūrinė diagrama pateikiama 7 pav.



7 pav. Reverberacijos efekto modeliavimo schema.

Iš diagramos išgauname lygtis [12-16], kurią išreiškiame Z ašyje. Viršutiniai nariai - a koeficientai, apatiniai - b.

$$y[n] = x[n] + K \cdot y[n - N] \tag{12}$$

$$y[z] = x(z)^0 + K \cdot y(z)^{-N} \tag{13}$$

$$H[z] = \frac{1}{1 - Kz^{-N}} \tag{14}$$

$$a = [1] \tag{15}$$

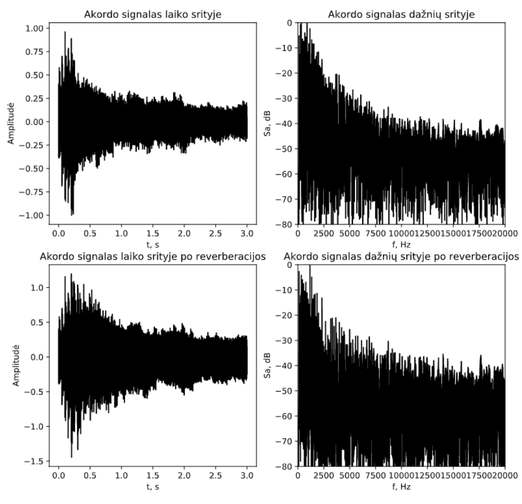
$$b = [1, 0, ..., 0, -K] \tag{16}$$

Tarp vieneto ir K kintamojo turime N-2 nulines reikšmes.

Eksperimentiniu būtų susiradome reverberacijos efektą gerai išreiškiantį vėlinimo koeficientą: N = 4400 atskaitų. K koeficientą pasirinkome 0.7, nes tai sukelia aukštesnės amplitudės reverberaciją, kuri geriau girdisi.

Žiūrint į 8 pav, laiko srityje galime pastebėti naujų aukštesnės amplitudės bangų signalo pradžioje, o žvelgiant į dažnių sritį - didesnį skirtingo decibelų lygio reprezentavimą kintant signalo dažniui. Reverberacijos efektas suteikė signalui naujų verčių pasekoje pradinio akordo.

Akivaizdu, kad K reikšmę palikus nuliu, signalas išliktų visiškai toks pat kaip prieš tai, o K=1 smarkiai moduliuotų signalą pagal reverberaciją.



8 pav. Gitaros akordo sumodeliuoto signalo laiko ir dažnių grafikai su reverberacijos efektu.

Amplitudinė ir žiedinė moduliacijos

Realizuojame du skirtingus moduliacijos metodus - amplitudinį bei žiedinį.

Amplitudinė moduliacija aprašoma lygtimi 17. Žiedinė moduliacija aprašoma lygtimi 18.

$$y[n] = (1 + \alpha \cdot \sin(2\pi n \frac{f_a}{f_d})) \cdot x[n] \tag{17}$$

$$y[n] = \sin(2\pi n \frac{f_m}{f_d}) \cdot x[n] \tag{18}$$

Eksperimentavimo būdu, galime pastebėti, kad amplitudinės moduliacijos atveju, α koeficientas įtakoja moduliacijos intensyvumą. Tą taip pat galime pastebėti iš formulės, jeigu α koeficientas yra parenkamas 0, tada mūsų sinusoidinė išraiška tiesiog neįtakos įeinančio signalo iš gausime y[n] = x[n]. Tačiau keliant α, stiprėja į reverberacijos efektą panašus efektas. Svarbų atkreipti dėmesį, kad vieneto pridėjimas šiuo atveju paverčia šį efektą savotiškai 'pridėtiniu', t.y jis tik pakoreguoja įeinantį signalą su papildomu skambesiu. Moduliavimo dažnis įtakoja šio efekto periodo ilgius. Mažesnis dažnis - ilgesni periodai.

Žiedinės moduliacijos atveju, tiesioginis įeinančio signalo dauginimas, nepridedant vieneto paverčia tai į visišką signalo transformaciją, kurios metu mes priverčiame savotišką aido efektą pradiniam akordui. α koeficiento neegzistavimas panaikina bet kokį papildomą slopinimo efektą, dėl to garso slopinimas tiesiogiai priklauso nuo įeinančio signalo.

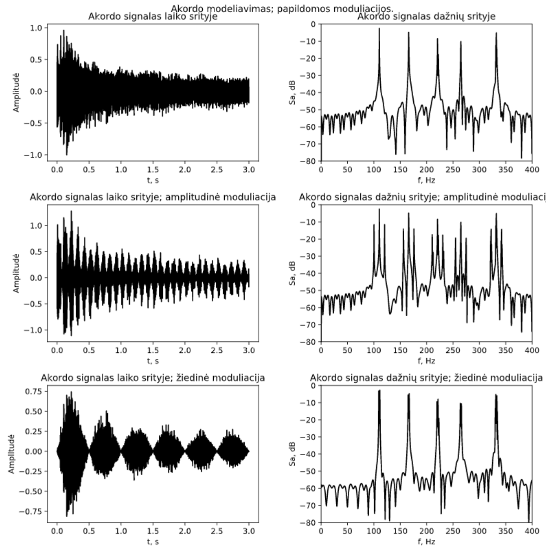
9 pav. matome moduliacijų efektus pradiniam akordo signalui. Šiuo atveju buvo naudojamos: $fm = 1hz, fa = 10hz, \alpha = 0.7$ koeficientų reikšmės. Amplitudinės moduliacijos atveju matome atsiradusį periodinį amplitudžių kitimą laike, o dažninė charakteristika parodo atsiradusias papildomas harmonikas tarp pagrindinių penkių. Žiedinės moduliacijos atveju, visas signalas radikaliai pakeistas į keletą intensyvių periodų, tačiau pačios harmonikos dažnių srityje stambiai nepakito, nes mes nepakeitėme pradinės harmonikos, o tiesiog pradėjom generuoji jos savotišką aidą.

Išvados

Šio laboratorinio darbo metu sumodeliavome atskirą gitaros stygą, pilna penkių stygų akordą ir pritaikėme įvairius signalo moduliacijos efektus.

Palyginus paprastos funkcijos mums padėjo išgauti itin tikroviškus garsus tokius kaip reverberacija ir aidas.

Taip pat pastebėjome, kad labai nedideli parametrų pakeitimai gali radikaliai pakeisti visą sugeneruotą signalą. Dažninė signalų charakteristika padeda pastebėti signalo pokyčius kurių galėjome nepastebėti iš laikinės diagramos. Galiausiai, pastebėjome kaip svarbu pažvelgti į signalus iš arčiau. Mūsų sumodeliuoto akordo atveju, žiūrint tik į pilnų trejų sekundžių laiko diagramą, galima lengvai praleisti stambius moduliavimo efektus, kurie išryškėja tik pažvelgus į amplitudžių svyravimus iš arti. Pavyzdžiui, iškraipymo efektas gali įnešti didelį triukšmo kiekį į signalą, kuris gali būti nepastebėtas žiūrint į signalą 'iš toli'.



9 pav. Gitaros akordo sumodeliuoto signalo laiko ir dažnių grafikai su papildomomis moduliacijomis.

Tik priartėjus pamatome, kaip amplitudžių peršokimo periodai gali prarasti tolydumą ir susidarę peršokimai sukelią triukšmo efektą.

Literatūra

[1] A. S. K. Karplus, Digital synthesis of plucked-string and drum timbres, *Computer Music Journal*, no. 7, pp. 43–55, 1983.

[2] numpy python skaičiavimų paketas. Nuoroda: https://numpy.org.

[3] scipy python mokslinių skaičiavimų paketas. Nuoroda: https://www.scipy.org.

[4] matplotlib python paketas. Nuoroda: $https : //matplotlib.org/api/pyplot_api.html.$

[5] scipy lfilter funkcija. Nuoroda: https://docs.scipy.org/doc/scipy/ reference/generated/scipy.signal.lfilter.html.

Priedai

Pagrindinės programos kodas

```
# To add a new cell, type '#␣%%'
# To add a new markdown cell, type '#␣%%␣[markdown]'
# %%
from IPython import get_ipython

null.tpl [markdown]
# # Laboratory work Nr.1 for the module T121M501 DSP
# # Number - 7
# # KTU 2020 Autumn Semester
# ## 1. Preparation
# ### 1.1 Resources
#
# [Markdown & LateX in jupyter notebooks (blog tutorial)](https://towardsdatascience.com/write-markdown-latex-in-the-jupyter-noteb
#
# [matplotlib.pyplot documentation (official api site)](https://matplotlib.org/api/pyplot_api.html)
#
# [Lab assignment document (onedrive share)](https://ktuedu-my.sharepoint.com/:b:/g/personal/luksta3_ktu_lt/EYmeVPJhKfVGoMwO6w6P8v
#
# [Main reference article for Karplus-Strong in Python](http://flothesof.github.io/Karplus-Strong-algorithm-Python.html)
#
# [Matlab doc for Karplus-Strong with filter()](https://www.mathworks.com/help/signal/ug/generating-guitar-chords-using-the-karplu

# %%
import numpy as np
get_ipython().run_line_magic('matplotlib', 'inline')
from numpy import cos, sin, pi, absolute, arange
import matplotlib.pyplot as plt
from scipy.signal import kaiserord, lfilter, firwin, freqz
from numpy import random
from scipy.ndimage.interpolation import shift
from IPython.display import Audio

null.tpl [markdown]
# # Specifics
#
# $
# \huge
# \begin{align}
# & 7 (Am) \\
# E (f_1) & = 0 \\
# A (f_2) & = 110 \\
# D (f_3) & = 165 \\
# G (f_4) & = 220 \\
# B (f_5) &= 262 \\
# e (f_6) &= 330
# \end{align}
# $
#

# %%
def karplus_strong_own_implementation(signal,n,delay):
   samples_passed = delay
   previous_value = 0
   while samples_passed < n:
      signal[samples_passed] = signal[samples_passed] + 0.5* (signal[samples_passed-delay] + signal[samples_passed-delay-1])
      samples_passed += 1
   return signal

def nir_filter_delay(fd,fs):
   return int(np.round(fd/fs))

def normalize_signal(x):
   x = x - x.mean()
   return x / (np.max(np.abs(x)))

def reverberation_filter_function(signal,N,K):
   # a = [1,0...0, -0.5,-0.5]
   # b = [1]
   a = np.concatenate(([1] , np.zeros(N-2) , [-K]),axis=0)
   b = [1]
   return lfilter(b,a,signal)

def karplus_strong_filter_function(signal,N):
   # a = [1,0...0, -0.5,-0.5]
   # b = [1]
   a = np.concatenate(([1] , np.zeros(N-3) , [-0.5,-0.5]),axis=0)
   b = [1]
   return lfilter(b,a,signal)

def string_generation(fs,fd,ts):
   # find out the amount of delay feedback steps required for signal generation
   N = nir_filter_delay(fd,fs)

   # generate random noise in the interval of [0,1] default
   noise = np.random.rand(int(N))

   # find out the amount of zero padding necessary for a full ts length signal
   zeros_count = fd * ts - N

   # generate the zero padding array
   zeros_padding = np.zeros((int(zeros_count)))

   # merge the noise and zero padding arrays
   signal = np.concatenate((noise,zeros_padding),axis=0)

   # filter the signal using the karplus strong algorithm and a delay of N
   filtered = karplus_strong_filter_function(signal,N)

   # normalize the filtered signal
   normalized = normalize_signal(filtered)

   return signal,filtered,normalized


def accord_generation(fs_array,fd,ts,delay_length):
   # initialize an array of zeros to fill with the generated signal
   signal = np.zeros(fd*ts)

   # initialize a value of total delay for each string
   total_delay = 0

   # set the amount of discrete delay points of a given delay length
   delay_samples = delay_length*fd

   # cycle all strings of the accord and add the signal to the total
   for fs in fs_array:
      # generate the signal
```

```python
        signal_0, filtered_0,normalized_0 = string_generation(fs,fd,ts)

        # shift it to simulate a delay
        normalized_0 = shift(normalized_0, total_delay, cval=0.0)

        # add the signal to our total
        signal += normalized_0

        # increase the amount of total delay for the next string
        total_delay += delay_samples

    # normalize the final signal
    signal = normalize_signal(signal)
    return signal


def convert_to_frequency_domain(signal):
    # generate a frequency domain array of the signal
    nfft = len(signal)
    S = np.abs(np.fft.fft(signal) / nfft)
    S = 20 * np.log10(S/np.max(S))
    k = np.linspace(0,nfft,nfft)
    f = k*fd/nfft
    return f,S

# %%
# 3.1.1 natos signalo modeliavimas

# a.) Randame N

fs = 165
fd = 44100
ts = 3

N = nir_filter_delay(fd,fs)

print(f"gavome {N} velinima D stygai\n")

# %%
# c.) sumodeliuojame D ir B stygas, pavaizduojame ju laiko ir dazniu vaizdus
# randame pirmuju triju harmoniku daznius. Aptarti pastebetus spektru panasumus ir skirtumus
fd = 44100

# D styga 165 Hz
d_fs = 165

d_n = nir_filter_delay(fd,d_fs)

d_signal, d_filtered,d_normalized = string_generation(d_fs,fd,ts)
d_f,d_S = convert_to_frequency_domain(d_normalized)

# B styga 262 Hz
b_fs = 262

b_n = nir_filter_delay(fd,b_fs)

b_signal, b_filtered,b_normalized = string_generation(b_fs,fd,ts)
b_f,b_S = convert_to_frequency_domain(b_normalized)

fig, axs = plt.subplots(2,2,figsize=(10,10),dpi=50)
#fig1, axs = plt.subplots(rows, cols, figsize=figsize, constrained_layout=True)

time_x = np.linspace(0,3,len(d_normalized))
axs[0,0].plot(time_x[d_n:d_n+3000],d_normalized[d_n:d_n+3000],'k')
axs[1,0].plot(time_x[b_n:b_n+3000],b_normalized[b_n:b_n+3000],'k')

#axs[0,0].plot(time_x,d_normalized,'k')
#axs[1,0].plot(time_x,b_normalized,'k')

axs[0,1].plot(d_f,d_S,'k')
axs[0,1].set_xlabel('f,␣Hz')
axs[0,1].set_ylabel('Sa,␣dB')
axs[0,1].set_xlim([0,1000])
#axs[0,1].set_xlim([0,fd/2])
axs[0,1].set_ylim([-80,0])

axs[1,1].plot(b_f,b_S,'k')
axs[1,1].set_xlabel('f,␣Hz')
axs[1,1].set_ylabel('Sa,␣dB')
axs[1,1].set_xlim([0,1000])
#axs[1,1].set_xlim([0,fd/2])
axs[1,1].set_ylim([-80,0])

axs[0,0].set_ylabel("Amplitude")
axs[1,0].set_ylabel("Amplitude")

axs[0,0].set_xlabel("t, s")
axs[1,0].set_xlabel("t, s")

axs[0,0].set_title("D stygos signalas laiko srityje")
axs[1,0].set_title("B stygos signalas laiko srityje")

axs[0,1].set_title("D stygos signalas dazniu srityje")
axs[1,1].set_title("B stygos signalas dazniu srityje")

fig.suptitle("D ir B stygu modeliavimas; apribotos X asys")

plt.show()

# TODO - harmoniku dazniai

# %%
# 3. Akordo signalo modeliavimas

# a.) pavaizduoti pilna akorda

fs_array = [110,165,220,262,330]
fd = 44100
ts = 3
delays = 0.05

accord_signal = accord_generation(fs_array,fd,ts,delays)
accord_f,accord_S = convert_to_frequency_domain(accord_signal)

fig, axs = plt.subplots(2,1,figsize=(10,10),dpi=50)
#fig1, axs = plt.subplots(rows, cols, figsize=figsize, constrained_layout=True)

time_x = np.linspace(0,ts,len(accord_signal))
```

```
axs[0].plot(time_x,accord_signal,'k')

accord_signal_reverberated = reverberation_filter_function(accord_signal,N,K)
#axs[0].plot(time_x[:3000],accord_signal[:3000],'k')

axs[1].plot(accord_f,accord_S,'k')
axs[1].set_xlabel('f,␣Hz')
axs[1].set_ylabel('Sa,␣dB')
#removable limit
axs[1].set_xlim([0,400])
axs[1].set_ylim([-80,0])


axs[0].set_ylabel("Amplitude")
axs[0].set_xlabel("t, s")

axs[0].set_title("Akordo signalas laiko srityje")
axs[1].set_title("Akordo signalas dazniu srityje")

fig.suptitle("Akordo modeliavimas; apribotos x asys")

plt.show()

Audio(accord_signal,rate=fd)

# %%


# %%
# Raskite akordo pirmuju penkiu harmoniku daznius. Ar sie dazniai sutampa su akordo atskiru stygu virpejimo dazniais?

K = [1,5,50]

def satlins(x):
   y = x
   for i in range(len(x)):
      if x[i] > 1:
         y[i] = 1
      elif x[i] < -1:
         y[i] = -1

   return y

# a.) pavaizduoti pilna akorda

fs_array = [110,165,220,262,330]
fd = 44100
fs = 3
delays = 0.05

accord_signal_base = accord_generation(fs_array,fd,ts,delays)
accords = []
accords_f = []
accords_S = []
for k in K:
   accord_signal = accord_signal_base * k
   accord_signal = satlins(accord_signal)
   accord_f,accord_S = convert_to_frequency_domain(accord_signal)
   accords.append(accord_signal)
   accords_f.append(accord_f)
   accords_S.append(accord_S)

#fig, axs = plt.subplots(len(K),2,figsize=(10,10),dpi=50)
#fig1, axs = plt.subplots(rows, cols, figsize=figsize, constrained_layout=True)
fig, axs = plt.subplots(len(K),2,figsize=(10,10),dpi=50,constrained_layout=True)
time_x = np.linspace(0,3,len(accord_signal_base))
for i in range(len(K)):

   axs[i,0].plot(time_x[:100],accords[i][:100],'k')
   axs[i,0].set_ylabel("Amplitude")
   axs[i,0].set_xlabel("t, s")
   axs[i,0].set_title(f"Akordo signalas laiko srityje, K={K[i]}")

   axs[i,1].plot(accords_f[i],accords_S[i],'k')
   axs[i,1].set_xlabel('f,␣Hz')
   axs[i,1].set_ylabel('Sa,␣dB')
   axs[i,1].set_xlim([0,400])
   axs[i,1].set_ylim([-80,0])
   axs[i,1].set_title(f"Akordo signalas dazniu srityje, K={K[i]}")

#fig.suptitle("Iskraipyto Akordo modeliavimas su skirtingomis K reiksmemis")
plt.show()

# %%
# 3. Reverberacijos modeliavimas

fs_array = [110,165,220,262,330]
fd = 44100
ts = 3
delays = 0.05

accord_signal = accord_generation(fs_array,fd,ts,delays)
accord_f,accord_S = convert_to_frequency_domain(accord_signal)

N = 4400
K = 0.7
accord_signal_reverberated = reverberation_filter_function(accord_signal,N,K)
accord_signal_reverberated = normalize_signal(accord_signal_reverberated)
accord_f_reverberated,accord_S_reverberated = convert_to_frequency_domain(accord_signal_reverberated)

fig, axs = plt.subplots(2,2,figsize=(10,10),dpi=50)
#fig1, axs = plt.subplots(rows, cols, figsize=figsize, constrained_layout=True)

time_x = np.linspace(0,ts,len(accord_signal))
axs[0,0].plot(time_x,accord_signal,'k')

axs[0,0].set_ylabel("Amplitude")
axs[0,0].set_xlabel("t, s")

axs[0,0].set_title("Akordo signalas laiko srityje")

#axs[0].plot(time_x[:3000],accord_signal[:3000],'k')

axs[0,1].plot(accord_f,accord_S,'k')
```

```
axs[0,1].set_xlabel('f,␣Hz')
axs[0,1].set_ylabel('Sa,␣dB')
#removable limit
axs[0,1].set_xlim([0,400])
axs[0,1].set_ylim([-80,0])
axs[0,1].set_title("Akordo signalas dazniu srityje")


axs[1,0].plot(time_x,accord_signal_reverberated,'k')
axs[1,0].set_ylabel("Amplitude")
axs[1,0].set_xlabel("t, s")
axs[1,0].set_title("Akordo signalas laiko srityje")
#axs[0].plot(time_x[:3000],accord_signal[:3000],'k')
axs[1,1].plot(accord_f_reverberated,accord_S_reverberated,'k')
axs[1,1].set_xlabel('f,␣Hz')
axs[1,1].set_ylabel('Sa,␣dB')
#removable limit
axs[1,1].set_xlim([0,400])
axs[1,1].set_ylim([-80,0])
axs[1,1].set_title("Akordo signalas dazniu srityje")



fig.suptitle("Akordo modeliavimas; apribotos x asys")
plt.show()
Audio(accord_signal,rate=fd)

# %%
Audio(accord_signal_reverberated,rate=fd)

# %%
fig, axs = plt.subplots(2,2,figsize=(10,10),dpi=50)
#fig1, axs = plt.subplots(rows, cols, figsize=figsize, constrained_layout=True)
time_x = np.linspace(0,ts,len(accord_signal))
axs[0,0].plot(time_x,accord_signal,'k')
axs[0,0].set_ylabel("Amplitude")
axs[0,0].set_xlabel("t, s")
axs[0,0].set_title("Akordo signalas laiko srityje")
#axs[0].plot(time_x[:3000],accord_signal[:3000],'k')
axs[0,1].plot(accord_f,accord_S,'k')
axs[0,1].set_xlabel('f,␣Hz')
axs[0,1].set_ylabel('Sa,␣dB')
#removable limit
axs[0,1].set_xlim([0,20000])
axs[0,1].set_ylim([-80,0])
axs[0,1].set_title("Akordo signalas dazniu srityje")


axs[1,0].plot(time_x,accord_signal_reverberated,'k')
axs[1,0].set_ylabel("Amplitude")
axs[1,0].set_xlabel("t, s")
axs[1,0].set_title("Akordo signalas laiko srityje po reverberacijos")
#axs[0].plot(time_x[:3000],accord_signal[:3000],'k')
axs[1,1].plot(accord_f_reverberated,accord_S_reverberated,'k')
axs[1,1].set_xlabel('f,␣Hz')
axs[1,1].set_ylabel('Sa,␣dB')
#removable limit
axs[1,1].set_xlim([0,20000])
axs[1,1].set_ylim([-80,0])
axs[1,1].set_title("Akordo signalas dazniu srityje po reverberacijos")



fig.suptitle("Akordo modeliavimas: reverberacijos efektas")
plt.show()

# %%
# 4. Papildomos moduliacijos
from copy import copy

def amplitude_modulation(a,fa,fd,x):
   y = copy(x)
   for i in range(len(x)):
      y[i] = (1 + a * np.sin(2*pi*i* fa/fd)) * x[i]
   return y

def ring_modulation(fm,fd,x):
   y = copy(x)
   for i in range(len(x)):
      y[i] = np.sin(2*pi*i* fm/fd) * x[i]
   return y
fs_array = [110,165,220,262,330]
fd = 44100
ts = 3
delays = 0.05
a = 0.7
fa = 10
fm = 1
accord_signal = accord_generation(fs_array,fd,ts,delays)
accord_f,accord_S = convert_to_frequency_domain(accord_signal)
N = 440
K = 0.7
accord_signal_amplitude_modulated = amplitude_modulation(a,fa,fd,accord_signal)
accord_f_amplitude_modulated,accord_S_amplitude_modulated = convert_to_frequency_domain(accord_signal_amplitude_modulated)
```

```
accord_signal_ring_modulated = ring_modulation(fm,fd,accord_signal)
accord_f_ring_modulated,accord_S_ring_modulated = convert_to_frequency_domain(accord_signal_ring_modulated)

fig, axs = plt.subplots(3,2,figsize=(10,10),dpi=50,constrained_layout=True)
#fig1, axs = plt.subplots(rows, cols, figsize=figsize, constrained_layout=True)

time_x = np.linspace(0,ts,len(accord_signal))
axs[0,0].plot(time_x,accord_signal,'k')

axs[0,0].set_ylabel("Amplitude")
axs[0,0].set_xlabel("t, s")

axs[0,0].set_title("Akordo signalas laiko srityje")

#axs[0].plot(time_x[:3000],accord_signal[:3000],'k')

axs[0,1].plot(accord_f,accord_S,'k')
axs[0,1].set_xlabel('f,␣Hz')
axs[0,1].set_ylabel('Sa,␣dB')
#removable limit
axs[0,1].set_xlim([0,400])
axs[0,1].set_ylim([-80,0])
axs[0,1].set_title("Akordo signalas dazniu srityje")


axs[1,0].plot(time_x,accord_signal_amplitude_modulated,'k')

axs[1,0].set_ylabel("Amplitude")
axs[1,0].set_xlabel("t, s")

axs[1,0].set_title("Akordo signalas laiko srityje; amplitudine moduliacija")

#axs[0].plot(time_x[:3000],accord_signal[:3000],'k')

axs[1,1].plot(accord_f_amplitude_modulated,accord_S_amplitude_modulated,'k')
axs[1,1].set_xlabel('f,␣Hz')
axs[1,1].set_ylabel('Sa,␣dB')
#removable limit
axs[1,1].set_xlim([0,400])
axs[1,1].set_ylim([-80,0])
axs[1,1].set_title("Akordo signalas dazniu srityje; amplitudine moduliacija")



axs[2,0].plot(time_x,accord_signal_ring_modulated,'k')

axs[2,0].set_ylabel("Amplitude")
axs[2,0].set_xlabel("t, s")

axs[2,0].set_title("Akordo signalas laiko srityje; ziedine moduliacija")

#axs[0].plot(time_x[:3000],accord_signal[:3000],'k')

axs[2,1].plot(accord_f_ring_modulated,accord_S_ring_modulated,'k')
axs[2,1].set_xlabel('f,␣Hz')
axs[2,1].set_ylabel('Sa,␣dB')
#removable limit
axs[2,1].set_xlim([0,400])
axs[2,1].set_ylim([-80,0])
axs[2,1].set_title("Akordo signalas dazniu srityje; ziedine moduliacija")


fig.suptitle("Akordo modeliavimas; papildomos moduliacijos.")

plt.show()

Audio(accord_signal,rate=fd)

# %%
Audio(accord_signal_amplitude_modulated,rate=fd)

# %%
Audio(accord_signal_ring_modulated,rate=fd)
```