SKAITMENINIS SIGNALŲ APDOROJIMAS IR MAŠININIS MOKYMASIS 2022

Laboratorinis darbas nr. 3

**ADAPTYVIŲJŲ FILTRŲ TYRIMAS**

**Ž. Marma, E MEI-2 gr.** Dėstytojas D. Sokas

*KTU, Elektros ir elektronikos fakultetas*

**Įvadas**

Adaptyvieji filtrai naudojami siekiant prisitaikyti prie kintančių triukšmo charakteristikų adaptuojant filtro koeficientus taip, kad jie duotuoju laiko momentu būtų optimalūs. Šios savybės neturintys filtrai su pastoviais koeficientais tampa neefektyvūs esant nestacionariam pašaliniam triukšmui. Todėl šiame darbe yra tiriami mažiausių vidutinių kvadratų (MVK), normalizuotų mažiausių vidutinių kvadratų (NMVK) ir dekoreliuotas rekursinio mažiausių vidutinių kvadratų algoritmai. Laboratorinio darbo tikslas – išmokti įgyvendinti ir tirti adaptyviuosius filtrus. Laboratorinio darbo užduotis yra adaptyviųjų filtrų pagalba prislopinti lėktuvo variklio triukšmo dedamąją piloto kalbos signale.

**Metodai**

**Adaptyvusis filtras**

Laboratorinio darbo schemoje (1 pav.) matyti, kad adaptyvusis filtras turi du įėjimus ir vieną išėjimą. Į vieną įėjimą patenka piloto kalbos 𝑠(𝑛) ir variklio triukšmo 𝑥(𝑛) sudėtinis signalas 𝑑(𝑛).Signalą 𝑑(𝑛) galima vadinti lėktuvo kabinos garsų signalu. Į kitą, atraminį įėjimą, patenka variklio triukšmo signalas 𝑥a (𝑛). Variklio triukšmo signalas turi būti registruojamas toje vietoje, kurioje nebūtų pašalinių garsų, o ypatingai – piloto kalbos. Reikia atkreipti dėmesį, kad signalai 𝑥(𝑛) ir 𝑥a(𝑛) yra tarpusavyje koreliuojantys, tačiau nevienodi, t.y. į piloto mikrofoną patenkantis variklio garso signalas būna pakitęs dėl perėjimo per kabinos sienas, kurias galima įsivaizduoti kaip tam tikrą filtrą.

Adaptyvusis algoritmas (blokas 5) kas kartą turi taip parinkti skaitmeninio filtro koeficientus (blokas 4), kad triukšmo įverčio signalas 𝑥įv(𝑛) taptų kuo panašesnis į triukšmo dedamąją, registruojamą kartu su piloto kalbos signalu. Tuomet iš kabinos mikrofono užregistruoto signalo 𝑑(𝑛) atėmus triukšmo įverčio signalą 𝑥įv(𝑛), gaunamas piloto kalbos signalo įvertis 𝑠įv(𝑛). Idealiu atveju, adaptyviuoju filtru išskirto piloto kalbos signalo įvertis turėtų būti identiškas triukšmu nepaveiktam piloto kalbos signalui 𝑠(𝑛).

Diagram

Description automatically generated

**1 pav.** Mažiausių vidutinių kvadratų adaptyviojo filtro schema piloto kalbos signalui išskirti.

Adaptyvusis filtras adaptuoja koeficientus tokiu būdu, kad klaida tarp tikrojo piloto kalbos signalo 𝑠(𝑛) ir jo įverčio 𝑠įv(𝑛) būtų minimali. Adaptyviojo algoritmo konvergavimo sparta ir stabilumas apsprendžiamas adaptacijos žingsniu 𝜇. Adaptacijos žingsnis įprastai parenkamas iš intervalo 0 < 𝜇 ≪ 1. Reikia pažymėti, kad didesnė adaptacijos žingsnio 𝜇 vertė leidžia greičiau adaptuoti skaitmeninio filtro koeficientus prie pasikeitusių signalo charakteristikų, tačiau parinktus per didelę 𝜇 vertę, algoritmas gali tapti nestabilus

**Mažiausių vidutinių kvadratų adaptyviojo filtro įgyvendinimas**

Filtro įgyvendinimas sudarytas iš šių etapų:

1. Pasirenkamos pradinės adaptyviojo filtro parametrų vertes – skaitmeninio filtro eilė 𝑀 ir adaptacijos žingsnis 𝜇.
2. Inicializuojami pradiniai filtro koeficientai. Filtro koeficientų vektoriaus narių skaičius atitinka filtro eilę. Pradinius filtro koeficientus patogu prilyginti nuliams (1)

|  |  |
| --- | --- |
| . | (1) |

1. Inicializuojamas atraminio įėjimo signalo vektorius. Atraminio įėjimo signalo vektoriaus narių skaičius atitinka filtro eilę. Pradinį atraminio įėjimo signalo vektorių taip pat sudaro nuliai (2)

|  |  |
| --- | --- |
| . | (2) |

1. Sudaromas atraminio įėjimo signalo vektorius 𝒙𝒂(𝑛). Vektorius yra sudarytas iš dabartinės signalo atskaitos n ir prieš tai buvusių signalo verčių. Atraminio įėjimo signalo vektoriaus narių skaičius atitinka filtro eilę M (3)

|  |  |
| --- | --- |
| . | (3) |

1. Variklio triukšmo signalo įvertis n-ajai atskaitai randamas atraminio įėjimo signalo vektorių nufiltravus koeficientais 𝐰(𝑛) (4)

|  |  |
| --- | --- |
|  | (4) |

1. Piloto kalbos signalo įvertis randamas iš kabinoje užregistruoto signalo atėmus variklio garso įverčio signalą (5)

|  |  |
| --- | --- |
|  | (5) |

1. Skaitmeninio filtro koeficientai 𝐰(𝑛 + 1), kurie yra naudojami diskretinio laiko momentu n + 1 yra atnaujinami pagal išraišką (6)

|  |  |
| --- | --- |
|  | (6) |

Klaidos kriterijumi dažnai naudojamas vidutinės kvadratinės klaidos 𝑀𝑆𝐸 (angl. *mean squared error*) įvertis, kuriuo įvertinama klaida tarp norimo signalo 𝑠(𝑛) ir adaptyviuoju filtru išskirto signalo įverčio 𝑠įv(n) (7)

|  |  |
| --- | --- |
|  | (7) |

**Normalizuotas mažiausių vidutinių kvadratų adaptyvusis algoritmas**

NMVK algoritmas yra įgyvendinamas analogiškai MVK algoritmui, tačiau skiriasi koeficientų adaptacijos principas. Adaptacijos koeficientas 𝜇 yra normuojamas pagal atraminio įėjimo signalo vektoriaus energiją (8)

|  |  |
| --- | --- |
|  | (8) |

**Dekoreliuotas rekursinis mažiausių kvadratų algoritmas**

MVK ir NMVK adaptyvūs algoritmai yra paprasčiau įgyvendinami, naudoja mažesnį matematinių operacijų skaičių, tačiau lėtai adaptuojasi, todėl gerai tinka tik stacionariems arba beveik stacionariems signalams filtruoti. Kur kas spartesniu konvergavimu pasižymi rekursinis mažiausių kvadratų (RMK) adaptyvusis algoritmas. Greitas RMK algoritmo konvergavimas pasiekiamas dėl koeficientų rekursijos, t.y. koeficientai, apskaičiuoti laiko momentu 𝑛 − 1, naudojami rasti koeficientus laiko momentu n.

**Rezultatai**

Šiame darbe buvo analizuojami trys (piloto, kabinos ir variklio triukšmo) 22 sekundžių trukmės garso signalai, diskretizuoti 8000 Hz diskretizavimo dažniu. Analizuojant signalus laiko srityje (2 pav.) galima pastebėti, kad kabinos signalas yra užterštas variklio triukšmu. Net ir klausantis garso įrašo sunku suprasti ką tiksliai sako pilotas.

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated with medium confidence

Chart, line chart

Description automatically generated

**2 pav.** Darbe analizuoti signalai laiko srityje

Analizuojant signalus dažnių srityje (3 pav.) galima pastebėti, jog variklio ir piloto signalai persidengia ties 400 – 1700 Hz dažniais.

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

**3 pav.** Darbe analizuoti signalai dažnių srityje

**Mažiausių vidutinių kvadratų algoritmo įgyvendinimas**

Įgyvendinant pradinį adaptyvųjį filtrą adaptacijos žingsnis buvo pasirinktas µ = 0,1 filtro eilė M = 20. Analizuojant piloto signalą laiko srityje (4 pav.) galima matyti, jog kalba buvo išskirta gerai. Klausantis garso įrašo tariami žodžiai yra suprantami, vienintelis trūkumas yra pašalinis garsas pradiniu laiko momentu (0-0,2 sekundė).

Chart, histogram

Description automatically generated

**4 pav.** Naudojant MVK algoritmą gautas piloto signalas laiko srityje

**Mažiausių vidutinių kvadratų algoritmo parametrų parinkimas**

Darbe buvo ištirta kokią įtaką filtro veikimui turi jo eilė – M. Eksperimentas buvo atliktas su pradiniu M = 20, dvigubai didesniu (M=40) ir dvigubai mažesniu (M=10). Adaptacijos žingsnis nekito ir buvo lygus µ = 0,1. Pastebima (5 pav.), kad esant mažesniam M signale lieka daugiau triukšmo dedamosios. Tačiau palygus signalus, kai M=20 ir M=40, galima matyti jog esant dvigubai didesnei filtro eilei išgautas piloto signalas neša daugiau informacijos. Remiantis tuo galime teigti, kad toliau didinant filtro eilė galime gauti nestabilų signalą (su labai didėlėmis reikšmėmis).

Chart, line chart

Description automatically generated

Chart, line chart, histogram

Description automatically generated

Chart

Description automatically generated

**5 pav.** MVK algoritmo palyinimas naudojant skirtingas M vertes

Įgyvendinto MKV algoritmo pagalba buvo rasti geriausi filtro parametrai. Filtro kokybė buvo įvertinta pagal MSE vertę (7). Ieškant optimalios filtro eilės ir adaptacijos žingsnio buvo analizuojamos šios vertės: M = [10, 15, 20, 25, 50, 100], 𝜇 = [0.001, 0.005, 0.01, 0.05, 0.1]. Gauti rezultatai yra pateikti grafiniu formatu (6 pav.). Pagal MSE rodiklį galima teigti, kad geriausia filtro parametrų pora yra M = 15 ir 𝜇 = 0,01. Galima matyti, jog didinant filtro eilę nebūtinai gaunami geresni rezultatai, kaip pavyzdžiui M=100, tačiau, kompiuteriniai resursai reikalingi skaičiavimams tikrai išauga.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

**6 pav.** MVK algoritmo filtro MSE priklausomybė nuo M ir 𝜇

Darbe buvo atlikta filtro kokybės pagal MSE analizė kintant laiku su skirtingomis adaptacijos žingsnio vertėmis (𝜇 = [0.001 0.01 0.1]). Analizės intervalas buvo pasirinktas 20ms, o filtro eilė M=15. Analizuojant rezultatus (7 pav.) galima pastebėti, kad mažiausią paklaidą pagal MSE rodiklį pasiekia filtras kurio 𝜇 = 0,01. Nors filtras su šiomis charakteristikomis ne taip greitai prisitaiko prie pradinio signalo kaip filtras naudojant 𝜇 = 0,1, tačiau prie tolimesnių signalo pokyčių sugeba geriau atlikti filtravimą.

Chart, histogram

Description automatically generated

Chart

Description automatically generated

**7 pav.** MVK algoritmo adaptyviojo filtro MSE priklausomybė nuo laiko su skirtingomis 𝜇 vertėmis

**Normalizuoto mažiausių vidutinių kvadratų adaptyviojo algoritmo įgyvendinimas**

Programiškai įgyvendinus NMVK algoritmą buvo suskaičiuota MSE vertę ir palyginta su MVK algoritmu (M=15, 𝜇 = 0,01). MSE MKV = 2.75∙10-4, MSE NMVK = 3.22∙10-4. Tiek įvertinus MSE rodiklius, tiek pasiklausius gautų piloto signalų galima teigti, kad algoritmų rezultatai yra labai panašūs. Pagal MSE rodiklį šio atveju filtras naudojantis MKV algoritmą būtų geresnis. Grafiškai analizuojant gautus signalus (8 pav.) galima matyti, kad vienintelis pastebimas skirtumas yra neženkliai išaugusi amplitudės reikšmė NMVK atveju.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

**8 pav.** MVK ir NMVK algoritmų palyginimas laiko srityje

**Dekoreliuotas rekursinis mažiausių kvadratų algoritmas**

Realizuojant rekursinį algoritmą pradines vertės buvo M = 15, λ = 0,99. Naudojant šias reikšmes MSE vertė gauta lygi 1,53 ∙10-3. Gautas piloto garso signalas (9 pav.) laiko srityje yra suprantamai panašus į gautus signalus naudojant ankstesnius algoritmus šiame darbe. Tačiau priešingai nei MVK ar NMVK pradiniu laiko momentu (0-0,2 s.) pastebimas itin staigus signalo amplitudės šuolis. Būtent šis laiko tarpas, kol algoritmas nesugeba prisitaikyti prie triukšmo pokyčių ir lemią didesnį MSE rodiklį.

Chart, line chart

Description automatically generated

**9 pav.** Piloto signalas naudojant RMK algoritmo adaptyvujį filtrą

Analogiškai MVK algoritmui RMK filtro algoritmui buvo rasti optimalūs parametrai M ir λ. Vertės buvo keičiamos taip: M = [10, 15, 20, 25, 50, 100], λ = [1, 0.99, 0.98, 0.97, 0.96, 0.95]. Grafiškai (10 pav.) galima matyti, jog didėjant užmiršimo konstantai MSE vertė mažėja. Geriausi rezultatai pasiekti naudojant M = 15 ir λ = 1. Šiuo atveju MSE vertė buvo 6,6∙10-4.

Chart, line chart

Description automatically generated

**10 pav.** RMK algoritmo filtro MSE priklausomybė nuo M ir λ

**Mažiausių vidutinių kvadratų RMK adaptyviųjų filtrų palyginimas**

Galiausiai MVK, NMVK ir RMK algoritmai buvo palyginti pagal MSE kitimą nuo laiko. Analizuojamų filtrų parametrai buvo pasirinkti atsižvelgiant į anksčiau gautas geriausias reikšmes: M = 15, λ = 1, µ = 0,01.

Remiantis MSE paklaidų kitimu nuo laiko (11 pav.) galima teigti, kad tinkamiausias algoritmas adaptyviajam filtrui yra RMK. Tačiau remiantis laboratoriniu darbo apraše esančiu MSE (1 lentelė) skaičiavimu geriausias algoritmas būtų MVK. Taip atsitinka, nes piloto signale gautu naudojant RMK pradiniu laiko momentu esanti paklaida iškreipia skaičiavimo tikslumą.

**1 lentelė.** Algoritmų palyginimas pagal MSE

|  |  |
| --- | --- |
| Metodas | MSE vertė |
| MVK | 2,75 ∙10-4 |
| NMVK | 3,22 ∙10-4 |
| RMK | 6,60 ∙10-4 |

Chart, histogram

Description automatically generated

Chart, line chart

Description automatically generated **11 pav.** RMK, MVK ir NMVK algoritmų palyginimas pagal MSE priklausomybė nuo laiko

**Diskusija**

Darbe buvo analizuojamas ganėtinai stacionarus triukšmo – variklio signalas. Įdomu būtų panagrinėti, kaip tokie adaptyviųjų filtrų algoritmai veikia su kintančiu motoro garsu – pvz. mašinoje keičiantis pavaroms. Galimai RMK algoritmo rezultatai būtų blogesni.

Adaptyviųjų filtrų vienas iš trūkumas yra pašalinis garsas pradiniu laiko momentu (0-0,2 sekundė).

RMK algoritmo apskaičiavimas užtruko ganėtinai ilgą laiką. Nors darbas buvo darytas su *Python* programa kuri vienu metu gali išnaudoti tik vieną centrinio procesoriaus branduolį, optimalių parametrų paieška užtruko gan ilgai. Į tai reikėtų atsižlėgti norit įgyvendinti šį algoritmą įterptinėje sistemoje su ribotais skaičiavimo resursais.

**Išvados**

Atlikus laboratorinį darbą buvo įsitikinta, kad kintančio triukšmo signalams filtruoti yra reikalingas adaptyvusis filtras.

Įgyvendinus MKV, NMKV ir RMK algoritmus galima teigti, kad iš esmės visi darbe nagrinėti algoritmai yra tinkami adaptyviajam filtrui.

Darbe buvo rasti geriausi parametrai MKV ir NVMK algoritmams: adaptacijos žingsnis μ=0.01 ir filtro eilė M=15. Ištyrus RMK algoritmą geriausi parametrai buvo rasti: M=15 ir λ = 1. Palyginus MKV ir NVMK algoritmus geriau triukšmą pašalina MKV algoritmas.

Darbe taip pat rastos MSE vertės algoritmams: MVK (2,75 ∙10-4), NMVK (3,22 ∙10-4) ir RMK (6,60∙10-4). Tačiau, vizualiai analizuojant MSE rodiklio grafiką buvo nustatyta, kad tinkamiausias yra RMK algoritmas.

**Literatūra**

Laboratorinio darbo aprašymas.

**Priedai**

Laboratorinio darbo *Python* programos kodas:

import scipy.io

import matplotlib.pyplot as plt

import numpy as np

from scipy.io import wavfile

from sklearn import preprocessing

class Adapt:

    def \_\_init\_\_(self, debug = False):

        self.debug = debug

        self.time\_s = 22

        self.Fs\_Hz = 8000

        self.saveDir = "out/"

    def drawSignal(self, signal\_y, title, show=False):

            tn = np.linspace(0, self.time\_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('Amplitudė')

            plt.grid(True)

          plt.savefig(self.saveDir + "amp\_" + title,  bbox\_inches='tight',

                        pad\_inches=0)

            if show:

                plt.show()

            plt.close()

    # 5 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.Fs\_Hz/nfft) for i in k]

        ax = plt.axes()

        ax.plot(f\_Hz, spectrum\_db)

        ax.set\_xlim(0, self.Fs\_Hz/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()

    def saveSignalAsWav(self, noteData, filename):

        wavfile.write(filename=filename, rate=self.Fs\_Hz, data=noteData.astype(np.float32))

# 3.4.1 MVK algoritmas

def calculate\_MVK(inNoise, inCombine, filterOrder=20, step = 0.1):

    if len(inCombine) != len(inNoise):

        print("Signals are not the same length")

        exit -1

    w   = np.transpose(np.zeros((1, filterOrder)))

    x\_a = np.transpose(np.zeros((1, filterOrder)))

    s\_iv = np.zeros((len(inNoise),))

    for n in range(len(inNoise)):

        x\_a = np.roll(x\_a, 1)

        x\_a[0] = inNoise[n]

        x\_iv = np.matmul(np.transpose(w), x\_a)

        s\_iv[n] = inCombine[n] - x\_iv[0]

        w = w + 2\*step\*s\_iv[n]\*x\_a

    return s\_iv

#3.6

def calculate\_NMVK(inNoise, inCombine, filterOrder=20, step = 0.1):

    if len(inCombine) != len(inNoise):

        print("Signals are not the same length")

        exit -1

    w   = np.transpose(np.zeros((1, filterOrder)))

    x\_a = np.transpose(np.zeros((1, filterOrder)))

    s\_iv = np.zeros((len(inNoise),))

    for n in range(len(inNoise)):

        x\_a = np.roll(x\_a, 1)

        x\_a[0] = inNoise[n]

        x\_iv = np.matmul(np.transpose(w), x\_a)

        s\_iv[n] = inCombine[n] - x\_iv[0]

        w = w + (step / (np.matmul(np.transpose(x\_a), x\_a))) \* s\_iv[n] \* x\_a # only this differs from MVK

    return s\_iv

def calculate\_RMK(inNoise, inCombine, filterOrder=15, Lam=0.99):

     if len(inCombine) != len(inNoise):

        print("Signals are not the same length")

        exit -1

    delta = 0.01

    I = np.eye(filterOrder)

    P = (delta\*\*-1) \* I

    w   = np.transpose(np.zeros((1, filterOrder)))

    x\_a = np.transpose(np.zeros((1, filterOrder)))

    s\_iv = np.zeros((len(inNoise),))

    for n in range(len(inNoise)):

        x\_a = np.roll(x\_a, 1)

        x\_a[0] = inNoise[n]

        v = np.matmul(P, x\_a)

        u = np.matmul(np.transpose(P), v)

        v\_norm = np.linalg.norm(v)

        k = 1 / (Lam + v\_norm\*\*2 + np.sqrt(Lam)\*np.sqrt(Lam+v\_norm\*\*2))

        P = (P - k \* np.matmul(v, np.transpose(u))) / np.sqrt(Lam)

        x\_iv = np.matmul(np.transpose(w), x\_a)

        s\_iv[n] = inCombine[n] - x\_iv[0]

        if np.isnan(s\_iv[n]):

            print(s\_iv[n])

        w = w + ((s\_iv[n] \* u) / (Lam + v\_norm\*\*2))

    return s\_iv

def calculate\_MSE(real, prediction):

    mse = (np.square(real - prediction)).mean()

    return mse

matData = scipy.io.loadmat('signalai/lab3\_signalai.mat')

print(matData)

print(type(matData))

variklioSig = matData.get('variklioSig')[0]

kabinosSig = matData.get('kabinosSig')[0]

pilotoSig = matData.get('pilotoSig')[0]

# # Normalize in 1/-1 range

# variklioSig = preprocessing.minmax\_scale(variklioSig, feature\_range=(-1,1))

# kabinosSig = preprocessing.minmax\_scale(kabinosSig, feature\_range=(-1,1))

# pilotoSig = preprocessing.minmax\_scale(pilotoSig, feature\_range=(-1,1))

print("Loaded signals")

print(type(kabinosSig))

plt.rcParams.update({'font.family': "Times New Roman"})

plt.rcParams.update({'font.size': 16})

plt.rcParams.update({'figure.figsize': (16, 6)}) # horizontally longer figure

filterObj = Adapt()

# Show signal over time and signal spectrum

if 1:

    # 3.3.1

    filterObj.drawSignal(variklioSig, "variklis")

    filterObj.drawSignal(kabinosSig, "kabinos")

    filterObj.drawSignal(pilotoSig, "piloto")

    filterObj.drawSpectrum(variklioSig, "variklis")

    filterObj.drawSpectrum(kabinosSig, "kabinos")

    filterObj.drawSpectrum(pilotoSig, "piloto")

    # 3.3.2

    filterObj.saveSignalAsWav(variklioSig, f"out/variklis.wav")

    filterObj.saveSignalAsWav(kabinosSig, f"out/kabina.wav")

    filterObj.saveSignalAsWav(pilotoSig, f"out/pilotas.wav")

    print("Saved wav files")

# 3.4 MVK

if 1:

    pilot\_afterMVK = calculate\_MVK(variklioSig, kabinosSig)

    filterObj.drawSignal(pilot\_afterMVK, "piloto po MKV")

    filterObj.drawSpectrum(pilot\_afterMVK, "piloto po MKV")

    filterObj.saveSignalAsWav(pilot\_afterMVK, f"out/pilotasMVK.wav")

    print("Calculated and saved signal after MKV")

# 3.5.6 - Compare with different M

if 1:

    M\_filterOrder = 20

    mu\_step = 0.1

    M\_array = [M\_filterOrder, M\_filterOrder\*2, M\_filterOrder//2]

    mu\_string = str(mu\_step)

    mu\_string = mu\_string.replace('.', ',')  # Convert for saving

    print("Analysis with different filter order")

    for M in M\_array:

        sig\_MVK = calculate\_MVK(variklioSig, kabinosSig, M, mu\_step)

        filterObj.drawSignal(sig\_MVK, f"piloto signalas, kai M = {M}, mu = {mu\_string}")

        filterObj.drawSpectrum(sig\_MVK, f"piloto signalas, kai M = {M}, mu = {mu\_string}")

        filterObj.saveSignalAsWav(sig\_MVK, f"out/piloto signalas, kai M = {M}, mu = {mu\_string}.wav")

# 3.5.7

# Filtro koeficientų skaičių 𝑀 galite keisti logaritminiu masteliu: 10,

# 20, 50, 100. Adaptacijos žingsnio 𝜇 vertę galite keisti dėsniu 0.001, 0.005, 0.01, 0.05, 0.1.

if 1:

    sig\_MVK = calculate\_MVK(variklioSig, kabinosSig)

    print(calculate\_MSE(pilotoSig, sig\_MVK))

    M = [10, 15, 20, 25, 50, 100]

    u\_mu = [0.001, 0.005, 0.01, 0.05, 0.1]

    print("Calculating MSE for different M and mu combinations")

    MSE\_array = np.zeros((len(M), len(u\_mu)))

    for i in range(len(M)):

        for j in range(len(u\_mu)):

            MSE\_array[i, j] = calculate\_MSE(pilotoSig, calculate\_MVK(variklioSig, kabinosSig, M[i], u\_mu[j]))

    print("Calculated MSE")

    print(MSE\_array)

    minIdx = np.unravel\_index(np.argmin(MSE\_array, axis=None), MSE\_array.shape)

    print(minIdx[0])

    print(minIdx[1])

    print(f"Min MSE is with: M={M[minIdx[0]]}, mu = {u\_mu[minIdx[1]]}")

    plt.figure

    for i\_miu in range(len(u\_mu)):

        temp\_mu = u\_mu[i\_miu]

        plt.plot(M, MSE\_array[:, i\_miu], marker='D', label=f"mu = {temp\_mu}")

    plt.title('MSE nuo M ir mu')

    plt.xlabel('M')

    plt.ylabel('MSE')

    plt.grid(True)

    plt.legend(loc="upper left")

    plt.savefig(filterObj.saveDir + "mse\_compare",  bbox\_inches='tight', pad\_inches=0)

    plt.show(block=False)

# plot MSE\_array over time

# Pavaizduokite adaptacijos greičio kreives (𝑀𝑆𝐸 priklausomybes nuo laiko) esant adaptacijos

# žingsnio vertėms 𝜇 = 0.001, 𝜇 = 0.01, 𝜇 = 0.1. 𝑀𝑆𝐸 galite skaičiuoti 10 ms ar ilgesniuose

# intervaluose.

if 1:

    intervalMSE\_s = 20 \* 10\*\*-3

    timeMSE\_ms = np.arange(0, filterObj.time\_s\*1000, intervalMSE\_s)

    bestM = 15

    test\_mu = [0.001, 0.01, 0.1]

    pilotEstimate1 = calculate\_MVK(variklioSig, kabinosSig, bestM, test\_mu[0])

    pilotEstimate2 = calculate\_MVK(variklioSig, kabinosSig, bestM, test\_mu[1])

    pilotEstimate3 = calculate\_MVK(variklioSig, kabinosSig, bestM, test\_mu[2])

    MSE\_overTime\_array = np.zeros((3, (len(timeMSE\_ms))))

    start\_i = 0

    end\_i = int(intervalMSE\_s\*filterObj.Fs\_Hz) # python needs int

    print("Calculating MSE over time")

    increaseInterval = end\_i

    for i in range(len(timeMSE\_ms)):

        MSE\_overTime\_array[0, i] = calculate\_MSE(pilotoSig[start\_i:end\_i], pilotEstimate1[start\_i:end\_i])

        MSE\_overTime\_array[1, i] = calculate\_MSE(pilotoSig[start\_i:end\_i], pilotEstimate2[start\_i:end\_i])

        MSE\_overTime\_array[2, i] = calculate\_MSE(pilotoSig[start\_i:end\_i], pilotEstimate3[start\_i:end\_i])

        start\_i = start\_i + increaseInterval

        end\_i = end\_i + increaseInterval

    plt.figure

    for i in range(len(test\_mu)):

        plt.plot(timeMSE\_ms, MSE\_overTime\_array[i], label=f"mu = {test\_mu[i]}")

    plt.title('MSE kitimas nuo laiko')

    plt.xlabel('t, s')

    plt.ylabel('MSE')

    plt.grid(True)

    plt.legend(loc="upper right")

    plt.savefig(filterObj.saveDir + "mse\_overtime\_mu",  bbox\_inches='tight', pad\_inches=0)

    plt.show(block=False)

# 3.6 - NMVK

# Normalizuoto mažiausių vidutinių kvadratų adaptyviojo algoritmo įgyvendinimas

if 1:

    best\_mu = 0.01

    bestM = 15

    print("Comparing MSE for MVK and NMVK ")

    pilot\_mvk = calculate\_MVK(variklioSig, kabinosSig, bestM, best\_mu)

    pilot\_nmvk = calculate\_NMVK(variklioSig, kabinosSig, bestM, best\_mu)

    filterObj.drawSignal(pilot\_mvk, "piloto naudojant MVK")

    filterObj.drawSignal(pilot\_nmvk, "piloto naudojant NMVK")

    mse\_mvk = calculate\_MSE(pilotoSig, pilot\_mvk)

    mse\_mvk\_mean = calculate\_MSE(pilotoSig, pilot\_nmvk)

    print(f"MSE of MVK: {mse\_mvk}")

    print(f"MSE of MVK mean: {mse\_mvk\_mean}")

# 4 Papildoma

if 1:

    sig\_RMK = calculate\_RMK(variklioSig, kabinosSig)

    filterObj.drawSignal(sig\_RMK, "piloto naudojant RMK")

    print(f"RMK MSE value: {calculate\_MSE(pilotoSig, sig\_RMK)}")

    M = [10, 15, 20, 25, 50, 100]

    lambda\_val = [1, 0.99, 0.98, 0.97, 0.96, 0.95]

    MSE\_array\_RMK = np.zeros((len(M), len(lambda\_val)))

    print("Started calculting MSE for RMK filter")

    for i in range(len(M)):

        for j in range(len(lambda\_val)):

            MSE\_array\_RMK[i, j] = calculate\_MSE(pilotoSig, calculate\_RMK(variklioSig, kabinosSig, M[i], lambda\_val[j]))

    print("Calculated MSE for RMK filter")

    minIdx = np.unravel\_index(np.argmin(MSE\_array\_RMK, axis=None), MSE\_array\_RMK.shape)

    print(f"Min MSE for RMK is with: M = {M[minIdx[0]]}, lambda = {lambda\_val[minIdx[1]]}")

    plt.figure

    for i\_lam in range(len(lambda\_val)):

        temp\_lam = lambda\_val[i\_lam]

        plt.plot(M, MSE\_array\_RMK[:, i\_lam], marker='D', label=f"lambda = {temp\_lam}")

    plt.title('MSE nuo M ir lamdos')

    plt.xlabel('M')

    plt.ylabel('MSE')

    plt.grid(True)

    plt.legend(loc="upper left")

    plt.savefig(filterObj.saveDir + "rmk\_diff\_lam", bbox\_inches='tight', pad\_inches=0)

    plt.show(block=False)

# compare all there metdods overtime

if 1:

    print("Comparing all 3 metdods")

    intervalMSE\_s = 40 \* 10\*\*-3

    timeMSE\_ms = np.arange(0, filterObj.time\_s\*1000, intervalMSE\_s)

    bestM = 15

    bestLambda = 1

    best\_mu = 0.01

    pilotEstimate1 = calculate\_MVK(variklioSig, kabinosSig, bestM, best\_mu)

    pilotEstimate2 = calculate\_NMVK(variklioSig, kabinosSig, bestM, best\_mu)

    pilotEstimate3 = calculate\_RMK(variklioSig, kabinosSig, bestM, bestLambda)

    MSE\_overTime\_array = np.zeros((3, (len(timeMSE\_ms))))

    start\_i = 0

    end\_i = int(intervalMSE\_s\*filterObj.Fs\_Hz) # python needs int

    increaseInterval = end\_i

    for i in range(len(timeMSE\_ms)):

        MSE\_overTime\_array[0, i] = calculate\_MSE(pilotoSig[start\_i:end\_i], pilotEstimate1[start\_i:end\_i])

        MSE\_overTime\_array[1, i] = calculate\_MSE(pilotoSig[start\_i:end\_i], pilotEstimate2[start\_i:end\_i])

        MSE\_overTime\_array[2, i] = calculate\_MSE(pilotoSig[start\_i:end\_i], pilotEstimate3[start\_i:end\_i])

        start\_i = start\_i + increaseInterval

        end\_i = end\_i + increaseInterval

    plt.figure

    plt.plot(timeMSE\_ms, MSE\_overTime\_array[0], label="MVK")

    plt.plot(timeMSE\_ms, MSE\_overTime\_array[1], label="NMVK")

    plt.plot(timeMSE\_ms, MSE\_overTime\_array[2], label="RMK")

    plt.legend(loc="upper right")

    plt.title('Algoritmu palyginimas')

    plt.xlabel('t, s')

    plt.ylabel('MSE')

    plt.grid(True)

    plt.ylim(0, 0.0025)

    plt.savefig(filterObj.saveDir + "Algoritmu palyginimas",  bbox\_inches='tight', pad\_inches=0)

    plt.show(block=False)