VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
KOMPIUTERINIO IR DUOMENŲ MODELIAVIMO KATEDRA

Signalų analizė ir apdorojimas - 2-oji užduotis

Cooley–Tukey greitosios Furje transformacijos algoritmas

Atliko:
Vita Mačinskaitė

Vadovas:
prof. Dr. Tadas Meškauskas

Vilnius
2021

1. Diskrečioji ir greitoji Furje transformacija

Šiame skyriuje trumpai pristatomi Diskrečiosios (DFT) ir Greitosios (FFT) Furje transformacijos algoritmai. Pasitelkus šiuos algoritmus visai nesunkiai kompiuterio pagalba galima analizuoti skaitmeninių signalų spektrus. Žemiau pateikta diskrečiąją Furje transformaciją aprašanti, visai paprasta formulė, kur f_j skaitmeninio signalo reikšmės, c_k po transformacijos gauto spektro komponentės.

$$c_k = 1/N \sum_{j=0}^{N-1} f_j e^{\frac{-i2\pi}{N} jk}, k = 0, 1, ..., N-1$$

Greitosios Cooley-Tukey transformacijos algoritmas yra vienas iš dažniausiai naudojamų FFT algoritmų, juo apdorojamo signalo reikšmių skaičius N turi būti lygus tam tikram dvejeto laipsniui $N = 2^n$, kur $n > 0$ yra sveikasis skaičius. Pagrindinė Cooley-Tookey algoritmo idėja yra suskaidyti DFT transformacijoje naudojamą sumą į dvi sumas, kuriose būtų tik signalo reikšmės su lyginiais arba nelyginiais indeksais. Šios sumos rekursiškai gali būti dalinamos tiek kartų, kol pradinė c_k suma bus išskaidyta į N sumų, kurių kiekvienoje tebus po vieną dėmenį, todėl liks sudėti tik N skaičių.

Šios užduoties metu pavyko suprogramuoti DFT ir FFT algoritmus. Šių algoritmų veikimo greitis buvo palygintas atlikus transformacijas su dirbtinai sugeneruotu, 8192 reikšmes turinčiu signalu. DFT algoritmas transformaciją įvykdė per 6 min 49 s, tuo tarpu FFT skaičiavimams užtruko 227 ms. Skirtumas akivaizdus. Suprogramuotas FFT algoritmas signalo transformaciją atliko net 1800 kartų greičiau. Didinant signalo reikšmių skaičių, šis skirtumas tik didėtų.

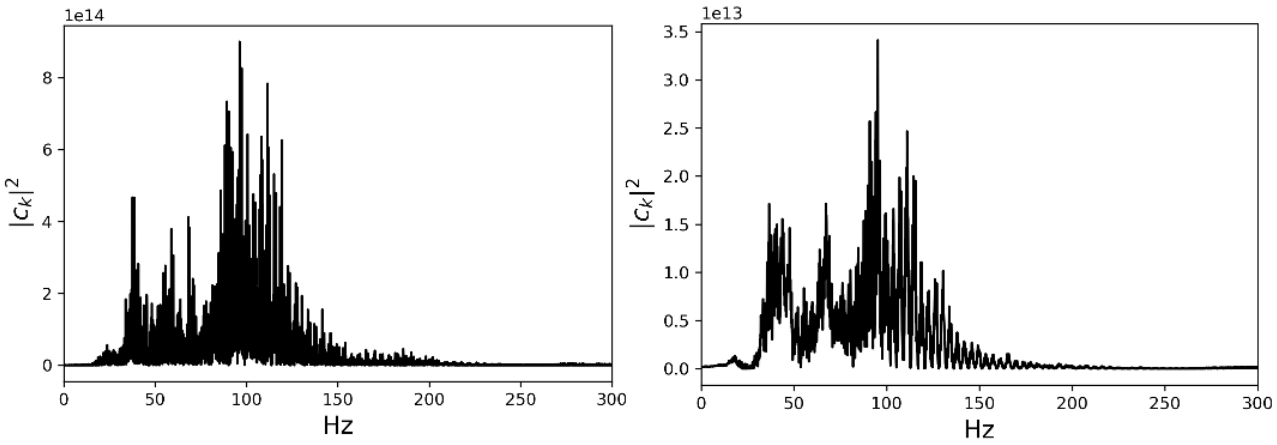
2. Furje filtrai

Šiame skyriuje aprašomi skirtingi Cooley Tookey algoritmo greitosios Furje transformacijos filtrai. Signalo komponenčių filtravimas slenkančio vidurkinimo metodu (žr. sk. 2.1), aukštų bei žemų dažnių užnulinimas (žr. sk. 2.2) ir signalo postūmis per tam tikrą hercų skaičių į aukštesnių (žr. sk. 2.3) ar žemesnių (žr. sk. 2.4) dažnių pusę bei jungtinis (poslinkio ir nulinimo) filtras (žr. sk. 2.5).

2.1. Signalo filtravimas taikant slenkantį vidurkį

Bene paprasčiausias FFT pritaikymo pavyzdys - c_k komponenčių vidurkinimas arba signalo glodinimas. Šiuo atveju bandyta atlikti garso signalo FFT reikšmių vidurkinimą. Šiam metodui išbandyti buvo pasirinktas širdies plakimo garso įrašas [4] (paklausyti galima *sirdiesPlakimasOrigin* faile). Prieš pradedant manipuliacijas su signalu, signalo reikšmių skaičius, pridedant nulius pakeistas į 2^n reikšmių signalą. Glodinimas buvo atliekamas apskaičiuojant pirmų 20 reikšmių vidurkį ir pastumiant vidurkinimo langą viena reikšme į priekį. Kadangi signalo spektras pasižymi simetrijos savybe, atitinkamai buvo vidurkinamos ir nuo $N/2$ iki N esančios reikšmės. Taikant šį metodą tikimasi, kad susumuotos, viena šalia kitos esančios komponentės eliminuos nežymius signalo registravimo triukšmus.

Atlikus auksčiau aprašytą vidurkinimą, gautas tylesnis garsas (galima paklausyti įrašo *SirdiesPlakimasVidurkintas*). Šio signalo galios spektrai prieš ir po vidurkinimo pavaizduoti pav. 1.



1 pav. Širdies plakimo įrašo glodinimas slenkančio vidurkio metodu. Kairėje signalas prieš, dešinėje po vidurkinimo

Kairėje pusėje pavaizduotas originalaus signalo galios spektras. Matoma, kad signalas turi daug triukšmo, žymesnės osciliacijos nėra pastebimos. Atlikus FFT ir po jos gautų komponenčių vidurkinimą, gautas dešinėje pusėje pavaizduotas grafikas. Signalas tapo daug švaresnis, matomos signalo osciliacijos. Tiesa toks signalo filtravimas šiam garso įrašui buvo naudingas tik vizualiai, stebint galios spektrus, pasiklausius garso įrašo, girdimas vis dar triukšmingas, tik tylesnis ir ne toks pastovus širdies plakimas.

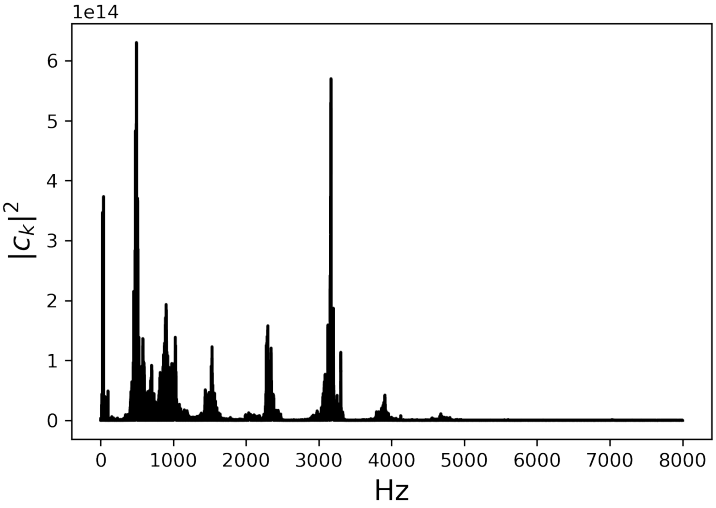
2.2. Šuns ir katės atskyrimas

Prieš taikant FFT algoritmą, du garsai [3] - šuns lojimas (įrašas - *dogOrigin*) ir katės miaukimas (įrašas - *catOrigin*) pakomponenčiui buvo sujungti į vieną garsą (įrašas - *catDog*). Prieš sujungimą signalo reikšmių skaičius buvo suvienodintas iki 2^n skaičiaus.

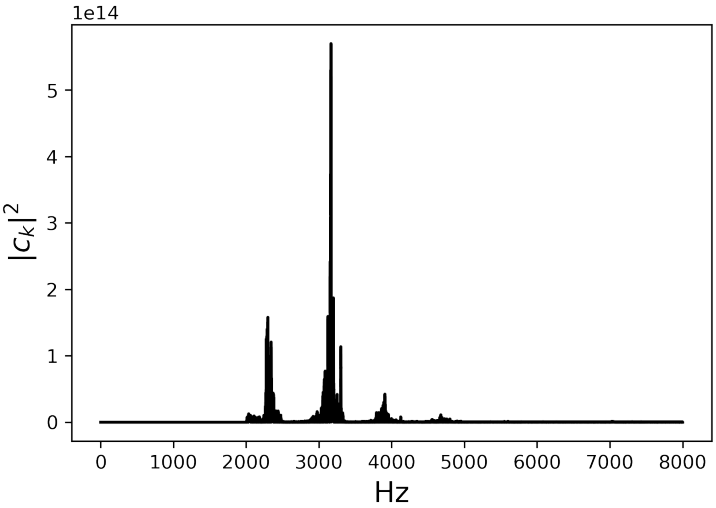
Atlikus FFT, gautosios signalo reikšmės c_k paverstos galios spektro koeficientais $|c_k|^2$. Signalo jėgos spektras pavaizduotas grafiku, jo x ašyje atidedant hercų (Hz) reikšmes. Hercai apskaičiuoti kiekvieno c_k koeficiento indeką k padalinus iš signalo trukmės sekundėmis. Gautas rezultatas pavaizuotas (2 pav.). Tokiu pačiu principu nubraižyti visi šiame darbe pavaizduoti signalų galios spektrai.

Signalas veikia 0-5000 Hz dažnio diapazone. Pagal internete rastą informaciją [2], šuns lojimas gali būti nuo 160 Hz iki 2630 Hz, priklausomai nuo priežasties, dėl kurios šuo loja. Todėl norint atskirti katės balsą nuo šuns, visos reikšmės iki 2000 Hz užnulinamos (3 pav.). Gautą rezultatą galima paklausyti pridėtame faile pavadinimu *noDogFilter*. Galime girdėti, kad šis filtras suveikė labai gerai, šuns lojimas girdimas tik kaip nedidelis krebždėjimas.

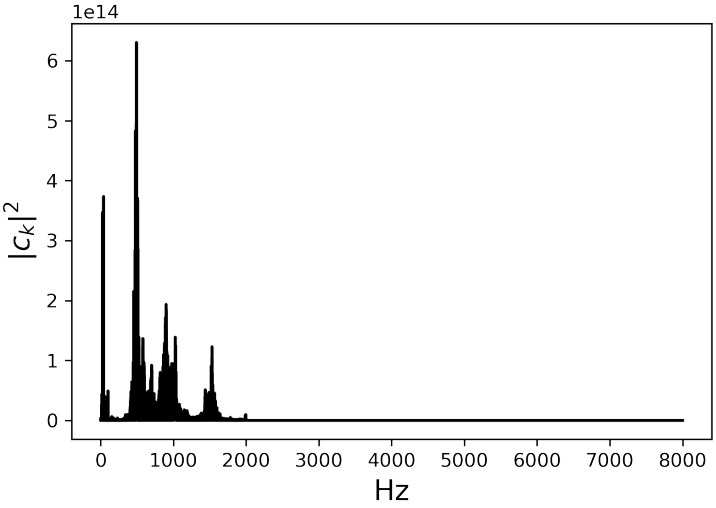
Kitas eksperimentas buvo atliktas užnulinant visus aukšto dažnio (2000-8000 Hz) garsus (4 pav.). Tikėtasi, kad įraše liks tik katės miaukimas. Įrašą galima paklausyti *noCatFilter* faile. Jame šiek tiek tyliau, bet vis dar aiškiai girdisi katės miaukimas, akivaizdu, kad jos balsas veikia ir žemų dažnių diapazone, todėl užnulinimo filtras šį kartą nesuveikė.



2 pav. Katės ir šuns signalo galios spektras



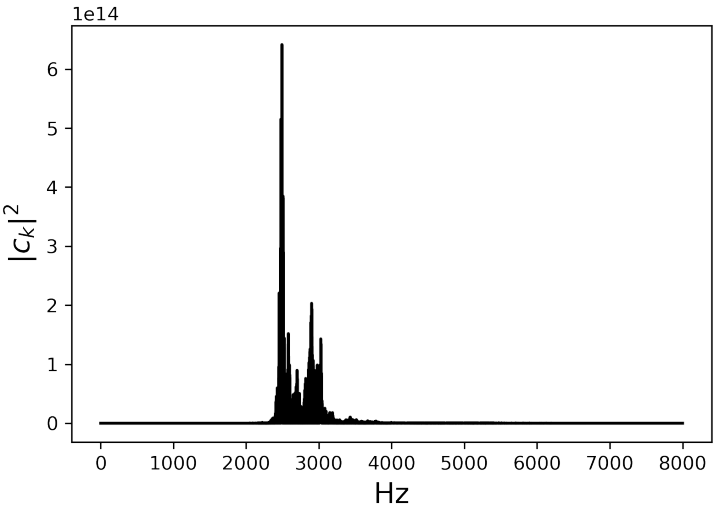
3 pav. Filtruotas katės-šuns signalo galios spektras, paliktas tik tariamai katės signalo dažnis (2000-8000 Hz)



4 pav. Filtruotas katės-šuns signalo galios spektras, paliktos tik žemų dažnių komponentės (0-2000 Hz)

2.3. Šuns pavertimas kate

Šiame skyriuje aprašomas filtro, paslenkančio FFT koeficientus į didesnių dažnių pusę veikimas. Norint suploninti šuns lojimą, arba šunį paversti kate, visas šuns lojimo signalas buvo pastumtas per 2000 Hz (žr. 5 pav.). Pirmiausia buvo surasta vartotojo nurodytą hercų kiekį atitinkanti galios spektro reikšmė. Pagal šios reikšmės indeksą, naudojant shift funkciją visos komponentės pastumtos į dešinę (aukštesnių dažnių pusę).

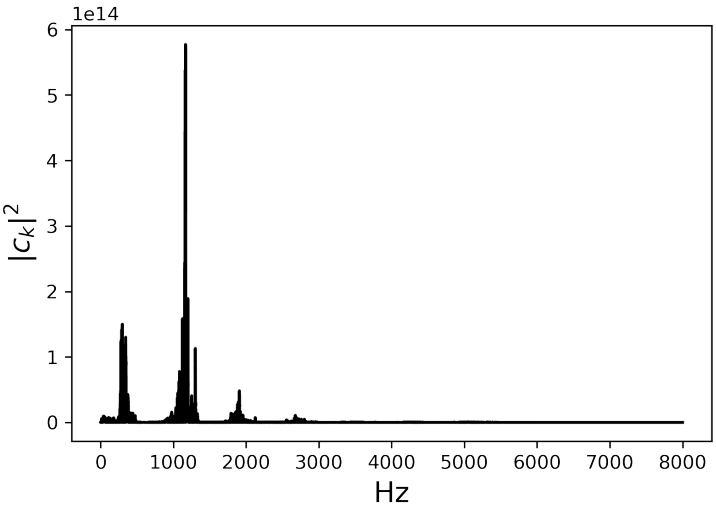


5 pav. Šuns lojimą vaizduojantis signalo galios spektras. Visos komponentės pastumtos į aukštesnių dažnių pusę per 2000 Hz

Gautą rezultatą galima išgirsti *pastumtasDog2000Hz* garso įraše. Šuo pavirto paukšteliu ar švilpuku, po filtravimo šio garso priskirti šuns lojimui tikrai nebegalima.

2.4. Katės pavertimas šunimi

Šiame skyriuje aprašomas filtro, paslenkančio FFT koeficientus į mažesnių dažnių pusę veikimas. Katės miaukimas buvo pažemintas pastumiant signalo komponentes per 2000 Hz (žr. 6 pav.). Pirmiausia buvo surasta vartotojo nurodytą hercų kiekį atitinkanti galios spektro reikšmė. Pagal šios reikšmės indeksą, naudojant shift funkciją visos komponentės pastumtos į kairę (žemesnių dažnių pusę).

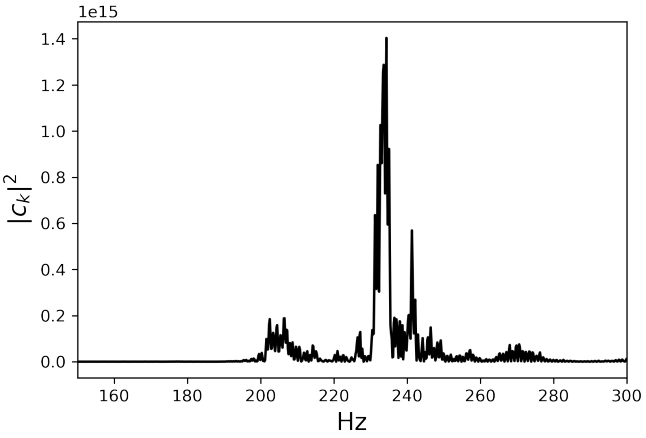


6 pav. Katės kniaukimą vaizduojantis signalo galios spektras. Visos komponentės pastumtos į žemesnių dažnių pusę per 2000 Hz

Gautą rezultatą galima išgirsti *pastumtasCat2000Hz* garso įraše. Katės miaukimas tapo žemesnis, tačiau pastūmus galios spektro reikšmes, dalis garso liko ganėtinai aukštų (2000 Hz) dažnių diapazone. Girdėti ir aukštesnis garsas.

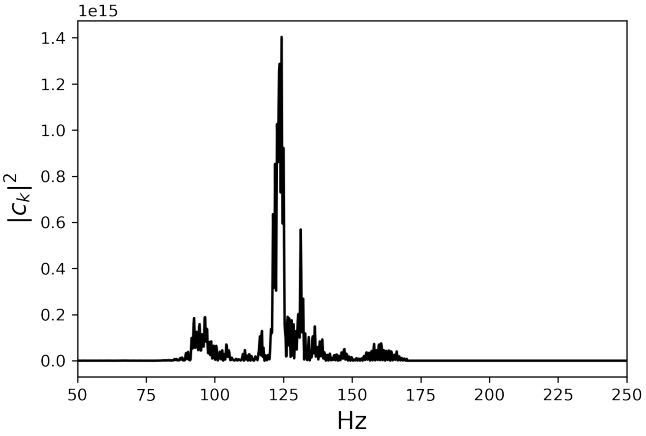
2.5. Mano balsas į vyrišką

Viena iš minčių atliekant šį darbą buvo pabandyti savo moterišką balsą paversti į vyrišką (originalus įrašas *manoBalsasLabaDiena*). Tam padaryti pritaikytas hercų pastumimo filtras į mažesnių dažnių pusę. 7 paveikslėlyje matoma, kad signalas pasiskirsto 200-280 Hz diapazone.



7 pav. Laba diena balso įrašo signalo galios spektras. Vaizduojamas originalus garsas.

Rasta informacija [1], kad vyriškas balsas gali svyruoti nuo 85Hz iki 180Hz. Todėl norint balsą padaryti vyrišku, visas spektras pastumtas į žemesnių dažnių pusę per 110Hz (balso įrašas *labadienaVyr110*). Girdėti, kad balsas tapo tikrai žemesnis. Tikintis gauti dar žemesnį balsą papildomai pašalintos visos spektro komponentės vyraujančios aukštesniame nei 170Hz diapazone (gautas garsas *labadienaVyr110nulintas*). Gautas garsas ne toks kokio tikėtasi, garsas daug tylesnis ir duslesnis, tarsi sklistų iš po vandens. Akivaizdu, kad nors ir nežymiai veikiančios, bet aukštesnio dažnio komponentės garsui suteikė aiškumo ir skaidrumo.



8 pav. Laba diena balso įrašo signalo galios spektras. Vaizduojamas filtruotas (pastumtas ir užnulintas) garsas.

Balsui paversti vyrišku, taikomas jugtinis Furje filtras, kur komponentės ne tik pastumiamos bet ir užnulinamos. Galutinis rezultatas matomas 8 paveikslėlyje.

Išvados

Šio tyrimo metu buvo suprogramuoti Diskrečiosios (DFT) bei Cooley-Tukey Greitosios (FFT) Furje transformacijos algoritmai.

FFT algoritmas sėkmingai pritaikytas signalų filtravimui.

Buvo išbandytos šios Furje filtro taikymo metodikos: FFT koeficientų vidurkinimas, sujungtų garsų atskyrimas šalinant aukštus bei žemus garsus, galios spektro pastūmimas į aukštesių bei žemesnių dažnių pusę bei jungtinis (pastūmimo ir dažnių šalinimo) filtras.

Darbas atliktas 2021 m. gegužės 28 d.

Literatūros bei signalų šaltiniai

[1] Informacija apie vyriškų balsų diapazoną.
https://www.quora.com/What-frequency-does-the-human-voice-have.

[2] Informacija apie šuns lojimo dažnį.
https://www.answers.com/Q/How_many_Hz_is_a_dog_bark.

[3] Katės kniaukimo ir šuns lojimo garsai.
https://www.kaggle.com/mmoreaux/audio-cats-and-dogs.

[4] Širdies plakimo garso įrašas.
https://bigsoundbank.com/.

Priedai

A. Programos kodo fragmentai

1 išeities kodas. DFT ir FFT

```
1
2 def mano_DFT(f, atvirkstine=False):
3     ck_reiksmes = []
4     N = len(f)
5     for j in range(0,N):
6         sum = 0
7         for k in range(0,N):
8             sum += f[k] * math.e ** ((1j if atvirkstine else -1j) *(2 *
                  math.pi/N) * j * k)
9         ck_reiksmes.append(sum/(N if atvirkstine else 1))
10    return ck_reiksmes
11
12 def mano_FFT(f_reiksmes, atvirkstine=False):
13    N = len(f_reiksmes)
14    if N == 1:
15        return f_reiksmes
16    W = np.exp((1j if atvirkstine else -1j)*2*np.pi/(N))*(1/N if
          atvirkstine else 1)
17    f_reiksmes_lygines = f_reiksmes[::2]
18    f_reiksmes_nelygines = f_reiksmes[1::2]
19    # vykdoma rekursija vis iskvieciant funkcija viduje
20    ck_lyginiai = mano_FFT(f_reiksmes_lygines)
21    ck_nelyginiai = mano_FFT(f_reiksmes_nelygines)
22    c = np.zeros(N, dtype=np.complex_)
23    for k in range(N//2):
24        c[k] = (ck_lyginiai[k] + W ** k * ck_nelyginiai[k])
25        c[k + N//2] = (ck_lyginiai[k] - (W**k) * ck_nelyginiai[k])
26    return c
```

2 išeities kodas. Funkcijos, naudotos filtravimui

```
1
2 def galios_spektras(fftr, trukme):
3     ckpow_reiksmes = abs(fftr)**2
4     N = len(ckpow_reiksmes)
5     Hercai = []
6     for k in range(0, N):
7         hercas = k/trukme
8         Hercai.append(hercas)
9     puse_hercu = Hercai[0:N//2]
10    ckpow_reiksmes_puse = ckpow_reiksmes[0:N//2]
11
12    plt.plot(puse_hercu, ckpow_reiksmes_puse, c='black')
13    plt.xlabel('Hz',fontsize = 15)
14    plt.ylabel('$|c_k|^2$', fontsize = 15)
15    plt.xlim(50, 250)
16    return plt.show()
17
18 # funkcija, kuri uznulina fft reiksmes simetriskai pagal hercu intervala
19 def uznulinti_Hz_intervale(fft_reiksmes, NulintinuoHz, nulintiikiHz):
20    N = len(fft_reiksmes)
21    Hercai = []
22    Hercu_indeksai = []
23    for k in range(0, N):
24        hercas = k/trukme
25        Hercai.append(hercas)
26    HercaiA = np.array(Hercai)
27    list_of_tuples = list(zip(Hercai, fft_reiksmes))
28    df = pd.DataFrame(list_of_tuples,
29                      columns = ['Hz', 'fft_reiksmes'])
30    nuliai = np.where(df['fft_reiksmes'] == 0)
31    if len(nuliai[0]) == 0:
32
33        df['fft_reiksmes'] = np.where(df['Hz'].between(NulintinuoHz,
              nulintiikiHz), 0, df['fft_reiksmes'])
34        nuliai1 = np.where(df['fft_reiksmes'] == 0)
35        df.loc[(len(fft_reiksmes)-len(nuliai1[0])):len(fft_reiksmes),'
              fft_reiksmes'] = 0
36        my_array = df['fft_reiksmes'].to_numpy()
37        return my_array
38    else:
39        return print('Error')
40
41 def slenkantis_vidurkis(reiksmes, poslinkis):
42    return np.convolve(reiksmes, np.ones(poslinkis), 'valid') /
          poslinkis
43
44 def shift(fft, postumis, fill_value=np.nan):
45    result = np.empty_like(fft)
46    if postumis > 0:
47        result[:postumis] = fill_value
48        result[postumis:] = fft[:-postumis]
49    elif postumis < 0:
50        result[postumis:] = fill_value
51        result[:postumis] = fft[-postumis:]
```

```
52      else:
53          result[:] = fft
54      return result
```