Sagedusruum

Sisukord

Aastal 2021 on vastutav juhendaja: Rasmus Saame <rasmus.saame@ut.ee> 2021-03-25|12:12:58 +0200

1. Ülevaade

Juhendist on olemas ka PDF versioon.

Selles praktikumis õpime sagedusruumi kohta ning vaatame kuidas läbi sagedusruumi konvolutsiooni teha. Selle jaoks teeme lähemalt tutvust kompleksarvude kasutamisega Pythonis ja implementeerime kiirema Fourier' teisenduse.

Juhendiga töötades pidage silmas

Praktikumijuhend sisaldab praktikumiülesannete kirjeldusi koos abistava infoga. Sealhulgas on toodud ka juhised oma süsteemi seadistamiseks. Töö õnnestumisele kaasa aitamiseks oleme teinud ka hulga märkmeid probleemsete kohtade osas. Selle info oleme jaganud tüübi järgi järgmise kolme kategooria vahel.

Siin toodud info võib aidata ülesande kiiremini või kavalamalt lahendada.

Siin toodud info võib aidata veaohtlikku kohta vältida.

Siin toodud info võib aidata vältida kahju riistvarale, tarkvarale või iseendale.

Töövahendid

Iga teegi järel on toodud klassi arvutites oleva paki versioon. See ei tähenda, et peate täpselt sama versiooni peale oma lahendusi kirjutama, küll aga tasuks jääda sama põhiversiooni piiresse ning, kui on valik, siis pigem klassis kasutatava versiooni kanti.

1. Python (3.8)
2. NumPy (1.20)
3. SciPy (1.6)
4. PyQtGraph (0.11)
5. Audacity (2.4)

Töö esitamine praktikumi lõppedes

Soovitame tungivalt hiljemalt iga alamülesande lõpetamise järgselt lisada uus versioon lahendusfailist oma harusse. GitLab keskkonnas on vaja luua *Merge request* ainult üks kord praktikumi jooksul.

1. Alustuseks veenduge, et olete oma kohaliku giti kausta **sees**
   a. Aktiivse kataloogi kontrollimiseks saate kasutada käsku `pwd` ning muutmiseks käsku `cd`
2. Kontrollige, et olete harul <**tudeng/eesnimi-perenimi**>
   a. Kasutage selleks käsku `git branch -l`
3. Sisestage käsk `git status`, et näha, kas on veel registreerimata muudatusi
   a. Kui mõni fail on muutunud, siis saate uue versiooni registreerimiseks kasutada käske `git add -p <muutunud-fail>` ja `git commit`
4. Kasutage käsku `git push`, et laadida kohalikud versioonid serverisse
   a. Lugege kindlasti käsu väljundit ning kahtluse korral kontrollige, kas failid jõudsid serverisse
5. Praktikumi lõppedes navigeerige veebilehele https://gitlab.ut.ee ning registreerige menüüde kaudu *Merge request* suunaga oma harust <**tudeng/eesnimi-perenimi**> harusse **master**
   a. Enne järgmist praktikumi vaatavad juhendajad teie pakutud versioonid üle ja liidavad need sobivuse korral master-nimelisse harusse

Enne alustamist

Repositooriumisse on lisatud uut sisu. Selle sisu kasutamiseks peate ta oma harusse liitma.

1. Alustuseks veenduge, et olete oma kohaliku giti kausta **sees**
   a. Aktiivse kataloogi kontrollimiseks saate kasutada käsku `pwd` ning muutmiseks käsku `cd`
2. Navigeerige master-nimelisele harule
   a. Kasutage selleks käsku `git checkout master`
3. Laadige alla ning liitke serveri versioon
   a. Kasutage selleks käsku `git pull`. Pange tähele ka käsu väljundina antud infot
4. Navigeerige tagasi oma harule
   a. Kasutage selleks käsku `git checkout tudeng/eesnimi-perenimi`. Muutke kindlasti käsus kasutatav haru nimi
5. Liitke kõik master-nimelisel harul olevad muudatused oma harule
   a. Kasutage selleks käsku `git merge master`

2. Kohustuslikud ülesanded

1. Ülesanne: Kompleksarvudega arvutamine

Sissejuhatus

Selle aine raames olete juba varasemalt kompleksarvudega kokku puutunud. Nüüd uurime natukene lähemalt, kuidas Python'is kompleksarvudega ümber käia ning tehteid teha. Selleks on Python'is sisseehitatud andmetüüp, millel on konstruktor kujul `complex(real, imag)`, mis võtab sisendiks arvu reaalosa ning imaginaarosa. Alternatiivselt saab kasutada kuju `z=a+bj`, näiteks võiks kirjutada `z=2+3j`.

Sellisel kujul salvestatud arvudega saab teha tehteid samamoodi nagu int või float tüüpi arvudega. Loodud kompleksarvust `z` on võimalik reaalosa ning imaginaarosa kätte saada kasutades isendimuutujaid (ehk isendivälju) `z.real` ning `z.imag`. Sama infot oskavad väljastada ka Numpy meetodid `np.real(z)` ning `np.imag(z)`. Kompleksarvu magnituudi on võimalik arvutada $\sqrt{Re(z)^2 + Im(z)^2}$, kus $Re(z)$ tähistab kompleksarvu reaalosa ja $Im(z)$ tähistab kompleksarvu imaginaarosa. Ka Python'i sisseehitatud funktsioon `abs(z)` tagastab kompleksarvulise sisendi korral arvu magnituudi.

Kompleksarvu $a + bj$ kaaskompleksiks nimetatakse arvu $a - bj$. Kaaskompleksidel on palju erinevaid kasulikke omadusi, millest ühega tutvume ka järgmises ülesandes.

Ülesande kirjeldus

1. Kompleksarvude liitmine ja lahutamine

   Esmalt vaatame kompleksarvude liitmist ja lahtumist. Leidke vastused järgmistele kompleksarvulistele tehetele. Arvutustes kasutage andmetüüpi `complex`. Olge valmis koodi juhendajale demonstreerima.

   $$(1 + 2j) + (3 - 4j) - (-2 - 0.4j) \quad (1)$$

   $$(-1) + (1j) + (1 + 2j) + (5 + 5j) + (-5 + 2j) \quad (2)$$

2. Kompleksarvude korrutamine ja jagamine

   Järgmisena vaatame kompleksarvude korrutamist ja jagamist. Leidke vastused järgmistele kompleksarvulistele tehetele. Arvutustes kasutage andmetüüpi `complex`. Olge valmis koodi juhendajale demonstreerima.

   $$\frac{(-5 - 4j) \cdot (9 + 7j)}{(-10 + j)} \quad (3)$$

   $$\frac{(-6 - 8j)}{(7 - 4j)} \cdot (-10 - 8j) \quad (4)$$

Seni oleme vaadanud avaldisi kujul $a + bj$, sellist kuju kutsutakse kompleksarvu algebraliseks kujuks. Kompleksarve saab aga esitada mitmel erineval kujul. Kompleksarvu eksponentkujuks

nimetatakse avaldist kujul $re^{j\theta}$, kus r on kompleksarvu magnituud ning θ on kompleksarvu faas. Eksponentkuju üks kasulik omadus on, et kompleksarvude korrutamine ning jagamine on mugav. Kahe eksponentkujul oleva kompleksarvu korrutise leidmiseks tuleb magnituudid omavahel korrutada ning faasid liita.

$$r_1e^{j\theta_1} \cdot r_2e^{j\theta_2} = (r_1 \cdot r_2)e^{j(\theta_1 + \theta_2)} \qquad (5)$$

3. Teisendamine eksponentkujule

   Implementeerige funktsioon `algebraic_to_exponent(c)`, mis teisendab kompleksarvu algebraliselt kuju ekspnentkujuks. Funktsioon peab võtma sisendiks kompleksarvu algebralisel kujul ning tagastama `tuple`(magnituud, faas). Selleks on abiks järgnev valem.

   $$a + bj = \sqrt{a^2 + b^2} \cdot e^{j * \theta} \qquad (6)$$

   Siin θ on kompleksarvu $a + bj$ faas. Teisendage $3 + 4j$ eksponentkujule.

   Ülesande lahendamisel ei tohi kasutada erinevate teekide funktsioone, mis on mõeldud eksponentkujule teisendamiseks, vaid tuleb ise implementeerida faasi ja magnituudi arvutamine.

4. Eksponentkujul korrutamine

   Implementeerige funktsioon `exponent_mult(r1, theta1, r2, theta2)`, mis võtaks sisendiks kaks kompleksarvu `tuple`(magnituud, faas) ning tagastaks nende korrutise kujul `tuple`(magnituud, faas). Oma funktsiooni saate testida järgneva tehte arvutamisega.

   $$1,5e^{j \cdot \frac{\pi}{2}} \cdot 5e^{j \cdot \frac{-\pi}{3}} \qquad (7)$$

5. Teisendamine algebralisele kujule.

   Implementeerige funktsioon `exponent_to_algebraic(c)`, mis võtab sisendiks kompleksarvu eksponentkujul ning tagastab kompleksarvu Pythoni `complex` andmetüübina. Selleks on abiks järgnev valem.

   $$r \cdot e^{j\theta} = rcos(\theta) + j \cdot rsin(\theta) \qquad (8)$$

   Teisendage algebralisele kujule $3e^{-j\pi}$.

6. Kompleksarvu n-astme juured

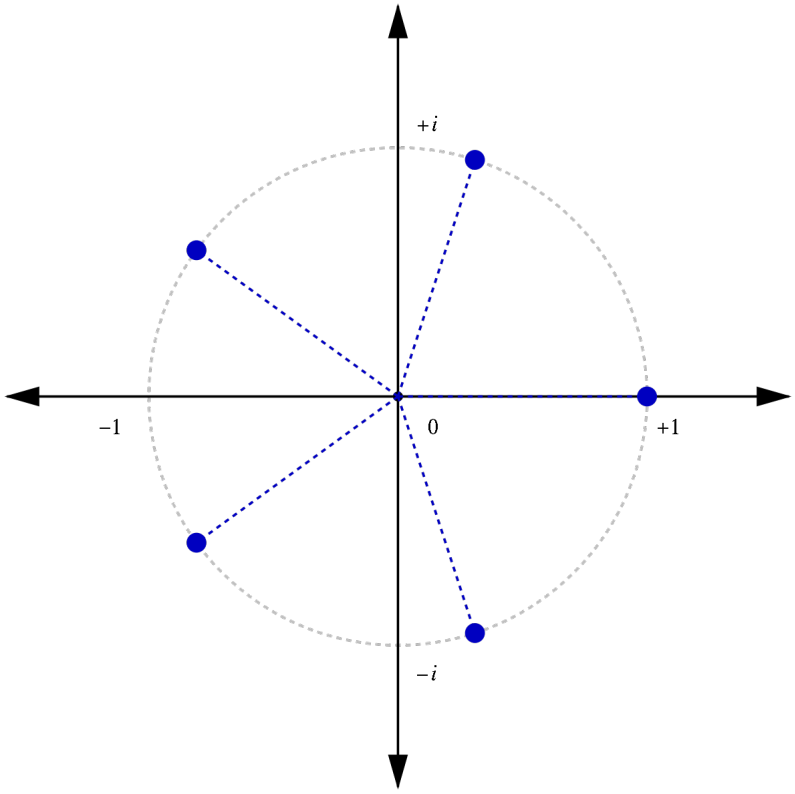
   Kompleksarvu n-astme juurteks nimetatakse kompleksarve, mida tõstes astmele n saame esialgse kompleksarvu tagasi. Igal kompleksarvul on täpselt n n-astme juurt.

   Implementeerige funktsioon `complex_roots(n)`, mis võtab sisendiks täisarvu n ning tagastab n -elemendilise massiivi võrrandi $z^n = 1$ kompleksarvulistest lahenditest, kasutades `complex` tüüpi. Kasutage selleks enda loodud `exponent_mult(r1, theta1, r2, theta2)` funktsionaalsust.

   Arvu 1 n-juured asuvad ühikringil võrdsete faasivahedega. Seega on võimalik juured eksponentkujul välja arvutada lihtsasti faasi muutes.

Arvu 1 n-astme juurtest saate juurde lugeda siit



Pildil on kujutatud arvu 1 kõik 5. astme juured. Näeme graafikult, et ühikringil paiknevad nad võrdsete vahedega.

Rohkem infot kompleksarvude kasutuse kohta Pythonis leiab siit.

Näidake oma lahendust praktikumi juhendajale ning tehke commit!

2. Ülesanne: Fourier' kiirteisendus

Sissejuhatus

Fourier' kiirteisendus (FFT) on algoritm, mis arvutab signaali diskreetset Fourier' pööret (DFT). Kuigi see saavutab täpselt sama tulemuse nagu eelmises praktikumis implementeeritud DFT algoritm, on see uskumatult tõhus, vähendades sageli arvutusaegu sadu kordi.

Võrreldes tavapärase DFT algoritmi keerukusega $O(n^2)$ on Fourier' kiirteisenduse keerukus $O(n \log n)$, kus n on sisendsignaali pikkus. Mida see tähendab? Ütleme, et mingi algoritm kuulub keerukusklassi $O(f(n))$, siis suurendades algoritmi sisendi suurust n-st m-ni, suureneb algoritmi täitmiseks kuluv aeg $\frac{f(m)}{f(n)}$ korda. Võrdleme, mitu korda rohkem aega kuluks DFT ja FFT algoritmil, kui sisendsignaali pikkus kasvab 1000 elemendi pealt 2000 elemendini. DFT algoritm kulutaks $\frac{2000^2}{1000^2} = 4$ korda rohkem aega, aga FFT algoritm kulutaks ainult $\frac{2000 \log(2000)}{1000 \log(1000)} \approx 2.2$ korda rohkem aega. See tähendab, et DFT muutus selle näite puhul ≈ 1.8 korda rohkem aeglasemaks, kui FFT. Suuremate näidete puhul on see vahe veelgi suurem. See erinevus ei pruugi tunduda suur, kuid FFT on andnud teadlastele ja inseneridele tõhusa viisi hüpata aja- ja sagedusruumi vahel, muutes mitmed lahendused võimatust võimalikuks.

Selles ülesande implementeerime Cooley-Tukey Fourier' kiirteisenduse algoritmi.

Kui te meenutate oma P05 DFT implementatsiooni, siis ilmselt mäletate, et soovitud tulemuse saavutamiseks pidite iga sageduse jaoks summeerima üle kõikide elementide. See on aga väga ajakulukas operatsioon. Alternatiivselt oleks saanud seda teha ühe suure maatriksiga korrutades, see aga ei ole keerukuse poolest efektiivsem.

Cooley-Tukey algoritm arvutab funktsiooni esituse Fourier' ruumis, kasutades vähem operatsioone, kui DFT. Esimese sammuna jagatakse sisendsignaal kaheks osaks: elemendid paarituarvuliste indeksitega ja elemendid paarisarvuliste indeksitega. Järgnevalt kordub rekursiivselt täpselt sama protseduur, kuni on tekkinud piisavalt lühikene järjend, millel sooritada Fourier' pööre. Kuna see algoritm jagab igal sisendil signaali 2 võrdse pikkusega alamsignaaliks, sobivad meie algoritmi sisendiks ainult signaalid, mille pikkus on esitatav kahe astmena. Cooley-Tukey poolt kirjeldatud algoritmile on aja jooksul leitud ka üldistusi. Tänu sellele on keerulisemate algoritmidega võimalik leida FFT'd mistahes pikkustega signaalidest.

Ülesande kirjeldus

See ülesanne jaguneb kolmeks loogiliseks tegevuseks. Esmalt implementeerite Fourier' kiirteisenduse ja seejärel implementeerite pöördteisenduse. Kolmas eristatav osa on visualiseerimine, mida on vaja mõlemas etapis toimuva näitamiseks. Proovige igal sammul kontrollida, kas teie väljund on õige. Kontrolliks kasutame signaali failist `sample_signal.py`.

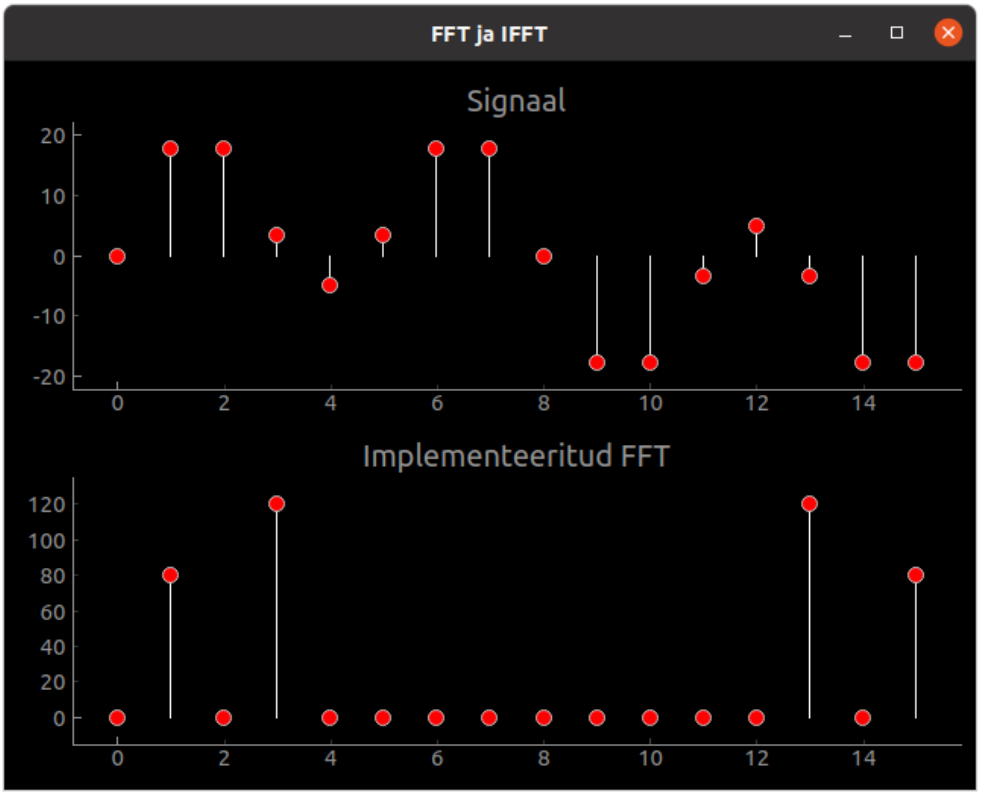
1) Implementeerige rekursiooni kasutades Cooley-Tukey Fourier' kiirteisenduse algoritm.

Teile on abiks aluskoodis asuv pseudokood.

1. Jagage sisendsignaal paarisarvulisteks ja paarituarvulisteks indeksiteks ning leidke mõlemast eraldi FFT. Siin jagame sisendsignaali kaheks jupiks ning toimub rekursiooni osa. Tänu rekursiooni sammule on FFT DFT-st palju efektiivsem.

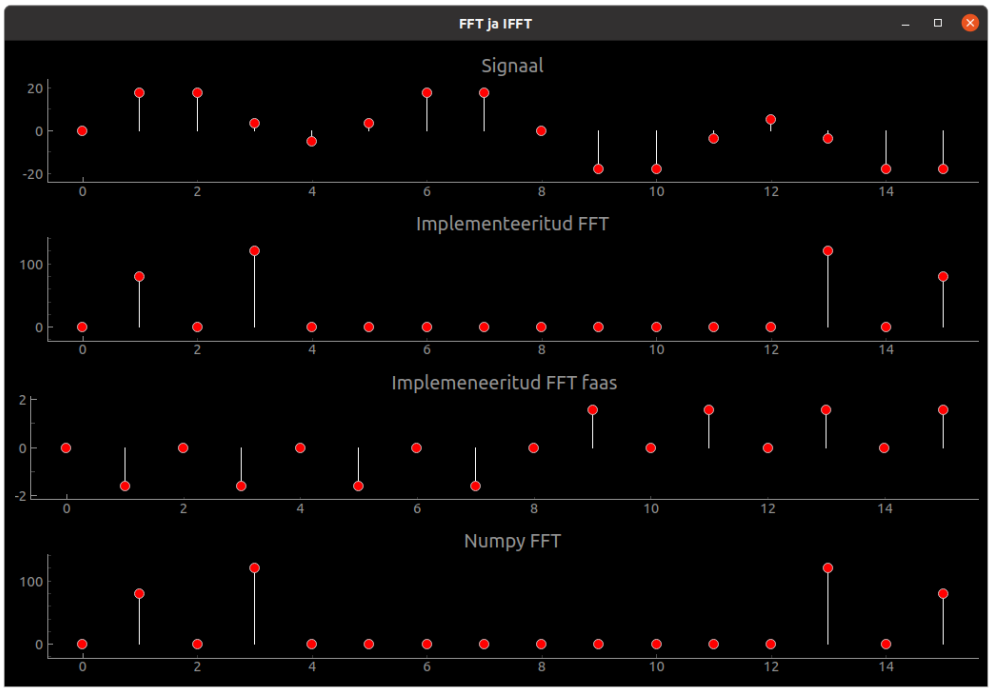
a. Juhul, kui signaali pikkus on täpselt 2, ei ole vaja FFT välja kutsuda.

2. Looge järjend [0, 1, 2, 3, ..., N-2, N-1]
3. Arvutusete lihtsustamiseks arvutage välja sisendsignaali pikkusest pool.
4. Tutvuge `twiddle` muutujaga. See massiiv sisaldab arvu 1 kõiki N-astme juuri. Kuidas oleks seda saanud arvutada teie loodud `complex_roots(n)` funktsiooniga?
5. Pange tähele, et see algoritm ei korruta väljundit arvuga $\frac{1}{N}$.



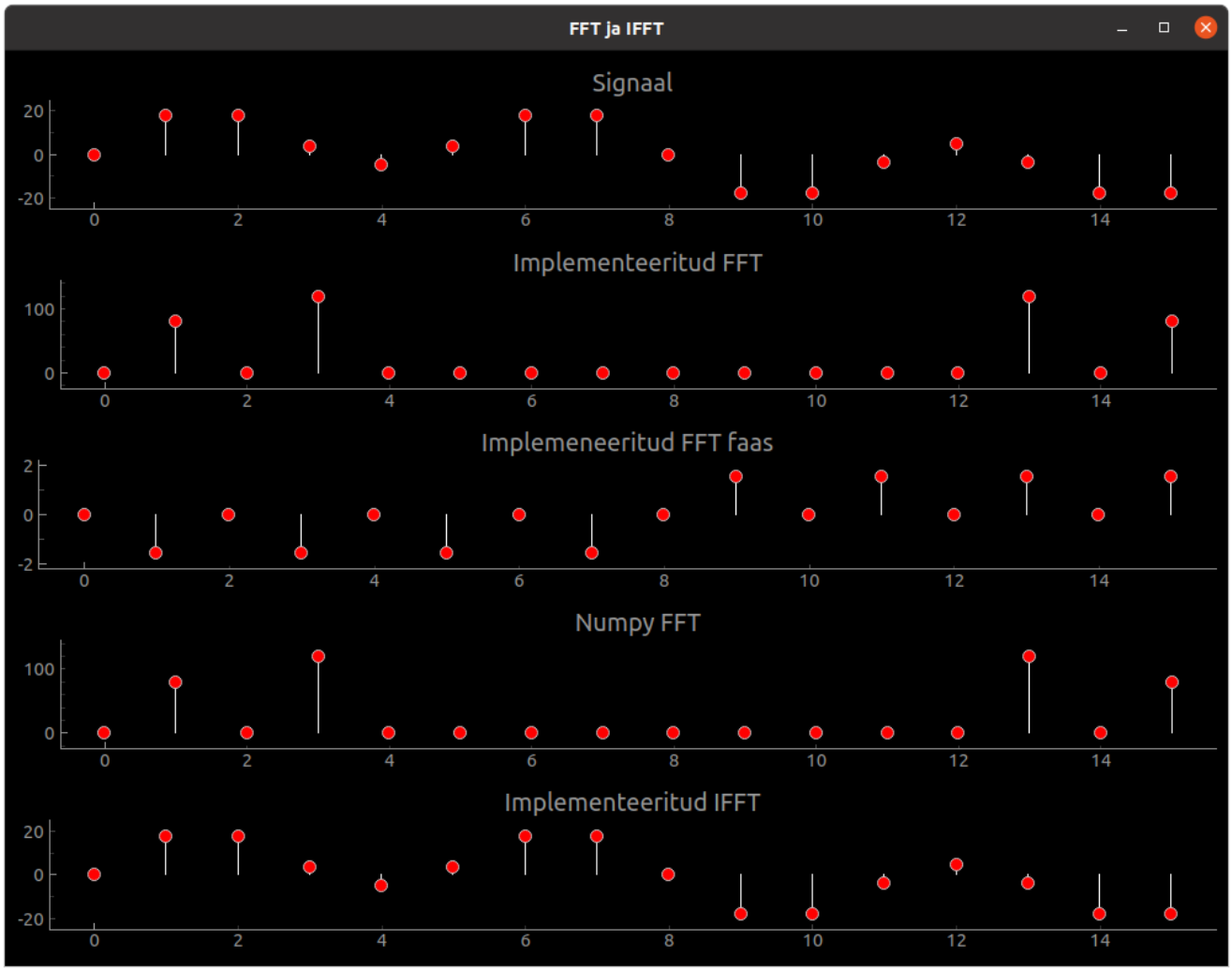
2) Looge veel graafikuid ning implementeerige IFFT

1. Võrrelge kiirteisenduse tulemust `numpy.fft.fft` tulemusega. Võrdluse lihtsustamiseks looge juba eelmisest praktilisest tööst tuttavad graafikud. Ka graafikute võrdluse juures tuleb jälgida, et võrreldaks kas sisendi pikkusega jagatud või jagamata tulemusi.
   a. Looge enda fft implementatsiooni näitamiseks veel üks graafik, näidake seal lisaks magnituudile ka fft väljundi faasi.



2. Implementeerige Cooley-Tukey Fourier' pöördteisendus eraldi funktsioonina. Selgub, et IFFT implementeerimiseks saab kasutada juba loodud FFT funktsionaalsust. Selleks...
    a. Leidke sisendsignaali kaaskompleks.
    b. Andke leitud väärtused sisendiks oma FFT funktsioonile.
    c. Leidke FFT väljundi kaaskompleks.
    d. Jätke alles kaaskompleksi reaalarvuline osa.
    e. Korrutage saadud reaalarvuline järjend arvuga arvuga $\frac{1}{N}$.
3. Võrrelge pöördteisenduse tulemust oma algse signaaliga kasutades `numpy.fft.ifft`. Looge taaskord abistavaid graafikuid.
4. Tulemuste graafilisel kujul esitamiseks meenutage, et:
    a. FFT puhul kujutame eraldi kompleksavu magnituudi ning faasi.
    b. Fourier' pöörde valemile leidub mitmeid erinevaid kujusid. Need kõik on õiged seni, kuni te neid segamini ei kasuta.

Lõpuks võiks teie graafik näha välja umbes selline:



Näidake oma lahendust praktikumi juhendajale ning tehke commit!

Esimene kodutöö

1. Koostage skeem enda teises ülesandes loodud FFT funktsiooni rekursiooni läbimisest. Sisendina kasutage signaali, milles on 8 punkti. Skeemi võib joonistada paberile ning tuleks sellisel juhul lisada fotona kodutöö faili ning viidata otselingiga täissuuruses failile, mis asub teie repositooriumis.

   Eeldatav vastus:

   - skitseerib skeemi / puu, kus on kujutatud andmepunktitde töötlemise järjekorda FFT algoritmis;
   - selgitab ühe kuni kahe lõiguga skeemi / puu ülesehitust toetudes enda koodile.

2. Mõõtke oma implementeeritud FFT ja P05 implementeeritud DFT algoritmi kiirust erinevate sisendite korral ning kujutage tulemusi graafikul.

   Eeldatav vastus:

   - sisaldab vähemalt viie erineva sisendiga sooritatud mõõtmiste tulemusi, kusjuures vähima ja suurima sisendi pikkuste vahe peab olema vähemalt tuhat korda;
   - võrdleb graafikul ning kirjeldab paari lausega, kui lähedal on teie implementatsioon esimeses praktikumiülesandes toodud keerukusele $O(n\log(n))$.

3. Ülesanne: Konvolutsioon läbi sagedusruumi
— *Overlap-add* —

Sissejuhatus

FFT konvolutsioon kasutab põhimõtet, et kahe signaali korrutamine sagedusruumis vastab konvolutsioonile ajas. Selle matemaatilist tõestust võib vaadata siit (b osa). Tegelikult lihtsalt sageduste korrutamisel saame tulemuseks ringkonvolutsiooni, kus konvolutsiooni otsad liituvad, meid huvitab aga lineaarne konvolutsioon. Sellest võimalusest oldi teadlikud juba Fourier' ajal, kuid see ei pakkunud laiemalt huvi. Nimelt Fourier' pöörde arvutamiseks kuluv aeg oli pikem otse konvolutsiooni arvutamiseks kuluvast ajast. See muutus 1965. aastal kui Cooley ja Tukey avalikustasid oma Fourier' kiirteisenduse algoritmi. Hilisem analüüs näitas, et sarnasele tulemusele olid erinevad matemaatikud jõudnud ka varem. Kasutades FFT algoritmi diskreetse Fourier' pöörde arvutamiseks saame lõpptulemuse läbi sagedusruumi kiiremini, kui kuluks konvolutsiooni arvutamiseks ajas. Lõpptulemus on sama, ainult tehte arv samasuguse tulemuse saamiseks on muutunud!

Varasemast praktilisest tööst võib meenuda, et teie poolt implementeeritud algoritm konvolutsiooni sooritamiseks ajas võttis juba lühikeste helifailide puhul väga palju aega. Selles ülesandes proovime näidata, et oma heliklippide FFT'ga sagedusruumi viimine, nende seal korrutamine ning tulemuse aega IFFT'ga tagasi toomine on kiirem kui konvolutsiooni tegemine ajas.

Et konvoleerida erineva pikkusega signaale läbi sagedusruumi, kasutame ositi konvoleerimise algoritmi (inglise keeles *overlap-add method*). Lihtsalt seletatuna on see sisendi jupi kaupa sagedusruumi viimine, seal impulsskostega korrutamine ning selle tagasi aegruumi toomine.

Alamülesanne 3.1: Sisendsignaali konvolutsioon läbi sagedusruumi:

Esmalt implementeerige 3. praktikumis tutvustatud sisendsignaali algoritmi, aga sedapuhku läbime ilmutatud kujul sagedusruumi. Jaotame teise signaali impulssideks ning iga impulsi jaoks arvutame ta mõju lõppsignaalile. Selleks paneme `s2_segment_length` muutuja väärtuseks 1.

Täiendage funktsiooni `overlap_add(s1, s2)` nii, et see leiaks kahe signaali konvolutsiooni läbi sagedusruumi. Eeldame, et `s1` on lühem signaal.

Selleks:

1. Viige esimene signaal sagedusruumi kasutades varasemalt loodud `fft` funktsiooni.
2. Algväärtustage väljundsignaal `result` nullidega. Kui pikk on kahe signaali konvolutsioon?
3. Tsükli sees…
    a. Viige teise signaali etteantud pikkusega lõik sagedusruumi oma `fft` funktsiooniga. Alamülesandes 3.1 on teile õige pikkusega lõigu loomine ette antud ning seda implementeerima ei pea, tuleb ainult lõik sagedusruumi viia
    b. Korrutage signaalide sageduskomponendid omavahel.
    c. Tooge tulemus tagasi aegruumi kasutades varasemalt loodud `ifft` funktsiooni.

Hetkel implementeeritud konvolutsioon ei ole aga kuidagi kiirem konvolutsioonist aegruumis, sest on endiselt $O(n^2)$ keerukusega. Tegelikult on 3.1 implementeeritud algoritm isegi aeglasem, sest lisaks kulub aega veel impulsside Fourier' pöördeks.

Alamülesanne 3.2: Overlap s2 lõikude vahel

Kui te võrdlete enda `overlap_add(s1, s2)` ning `numpy.convole(s1, s2)` konvolutsioonide tulemusi, näete, et need on identsed. Siiski ei ole meie implementatsioon konvolutsioonist läbi sagedusruumi veel kaugeltki valmis. Mis juhtuks, kui me üritaksime `overlap_add(s1, s2)` esimeseks sisendiks anda signaali, mille pikkus ei oleks kujul 2^n?

Järgmisena lisame oma lahendusele võimaluse kasutada sisendina ka signaale, mille pikkus ei ole kujul 2^n, ning vaatame korraga pikemaid lõike `s2`-st. Pikemate lõikude vaatamine vähendab meie tsükli läbimiste arvu mitmekordselt ning algoritm muutub sellevõrra kiiremaks (terve keerukusklassi jagu).

Hetkel piirab meie `s2` vaadeldava lõigu pikkust muutuja `convolve_segment_length`, mis tähistab seda, kui pikk on meie `s1` ja `s2` lõigu konvolutsiooni pikkus. Seega, et suurendada `s2` vaadeldava lõigu pikkust, suurendame muutujat `convolve_segment_length`. Muutuja `convolve_segment_length` pikkus peab kindlasti olema kahe aste, sest vastasel juhul ei sobi ta meie FFT algoritmi sisendiks. Samuti peab ta olema `s1` pikkusest suurem. Et vältida alamülesandes 3.1 juhtunud olukorda, et `s2` läbiti väga väikeste juppide kaupa, vaatame hoopis järgmist kahe astet.

1. Arvutage välja `convolve_segment_length` suurus, kasutades `s1` pikkust.

Järgmiseks peame `s1` viima sagedusruumi. Eelnevalt näitasime, et on kasulik, kui `s1` on pikkusega `convolve_segment_length`, lisame selle saavutamiseks `s1` piisavalt 0 kasutades näiteks funktsiooni numpy.pad

2. Viige `s1` sagedusruumi.

Nüüd saame välja arvutada, kui pikka lõiku saame me korraga vaadata `s2`. Mis peaks olema muutuja `s2_segment_length` pikkus, et `s1` ja `s2`-st võetud `s2_segment_length` pikkusega lõigu konvolutsiooni pikkuseks peab olema `convolve_segment_length`?

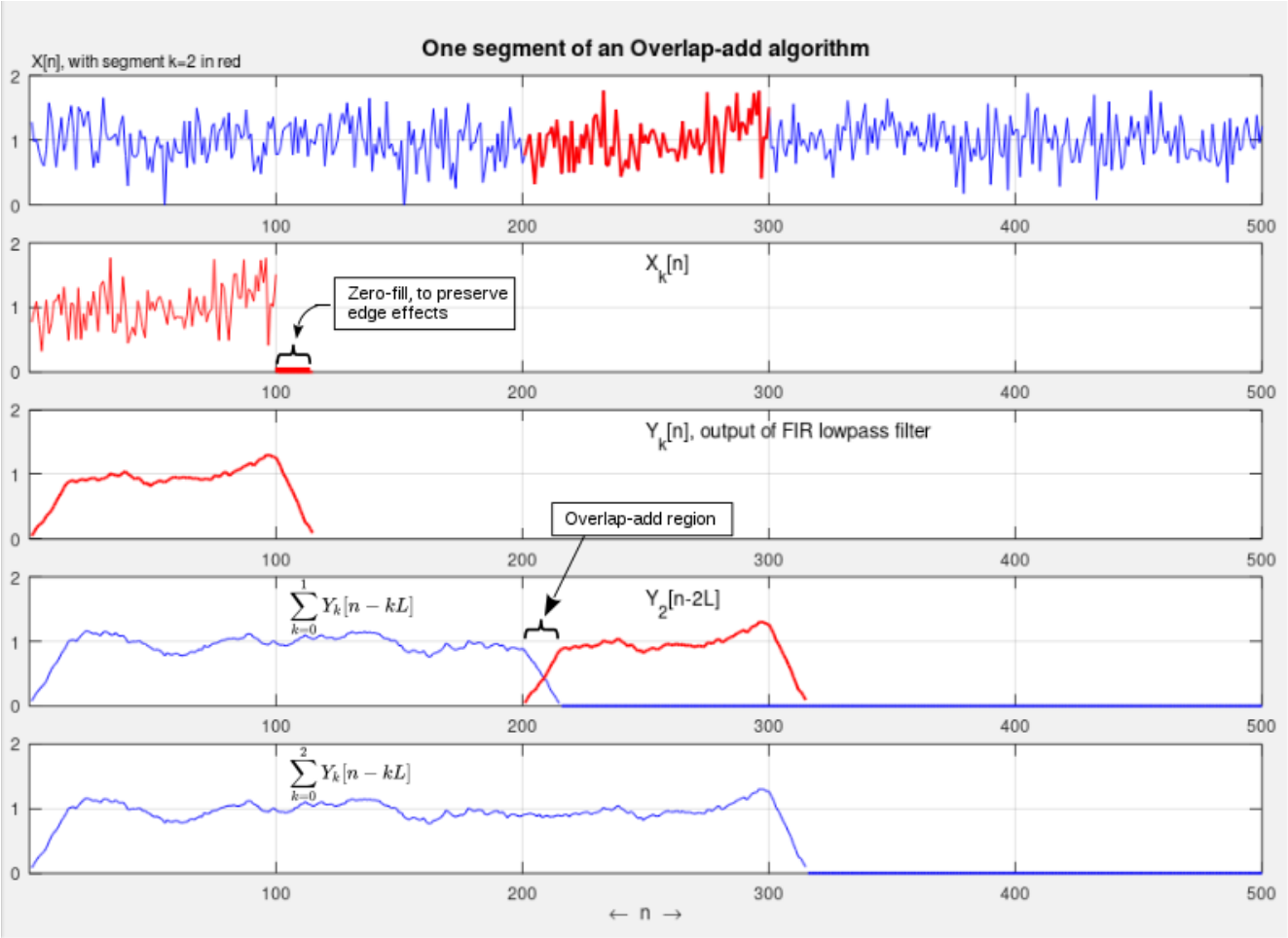
3. Arvutage välja `s2_segment_length` suurus, kasutades `s1` pikkust ning `convolve_segment_length` 'i.
4. Tsükli sees…
    a. Viige lõik `s2`-st sagedusruumi.

       Võtame `s2` lõigu, mis oleks pikkusega `s2_segment_length`. Et lõiku sagedusruumi viia `fft` funktsiooni abil, peab ta pikkus olema `convolve_segment_length`. Lisage leitud lõigule piisavalt 0 taha, et seda saavutada.

    b. Korrutage sagedusruumis signaalid ning tooge tulemus tagasi aegruumi.
    c. Liitke konvolutsiooni tulemus tulemussignaalile.

Üritades liita leitud lõigu konvolutsiooni tulemussignaalile, võib tekkida olukord, kus leitud konvolutsioon on liiga pikk ning ei mahu tulemussignaali ära. Sellisel juhul tuleb liita ainult nii palju kui mahub.

Kui te olete teinud kõik korrektselt, peaks teie *Overlap-add* töötama igasuguste signaalidega tingimusel, et esimene sisendsignaal on teisest lühem.



Alamülesanne 3.3 Helifailide peal katsetamine

Peale *overlap-add* algoritmi implementeerimist teostage FFT konvolutsioon järgnevate sammudega:

1. Lugege sisse kaks helifaili (sisendheli ja impulsskoste) ning muutke järjendite andmetüüp vormi *float32*.
    a. Uurige np.astype funktsionaalsust.
2. Konvoleerige signaalid
    a. Kasutage selleks enda poolt loodud `overlap_add` funktsiooni.
3. Normaliseerige oma konvolutsiooni tulemus $[-1, 1]$ vahele ning kirjutage see faili.

Arvestuse saamiseks konvoleerige `guitar_very_short.wav` (P06 kaustas) ühe impulsskostega (P03 kaustas) kasutades oma poolt kirjutatud `overlap_add` funktsiooni. Konvoleerige ka samad signaalid ajas, kasutades 3. praktikumis enda poolt implementeeritud konvolutsiooni algoritmi. Mõõtke aeg, mida kumbki meetod oma tööks kulutab ning kontrollige mõlemat konvolutsiooni väljundfaili

Audacity’s.

Veenduge, et mõlema faili sämplimissagedus oleks 8kHz, et P03 implementeeritud konvolutsioon mõistliku ajaga töö lõpetaks.

Näidake oma lahendust praktikumi juhendajale ning tehke commit!

4. Ülesanne: Sagedused ja sagedusvahemikud

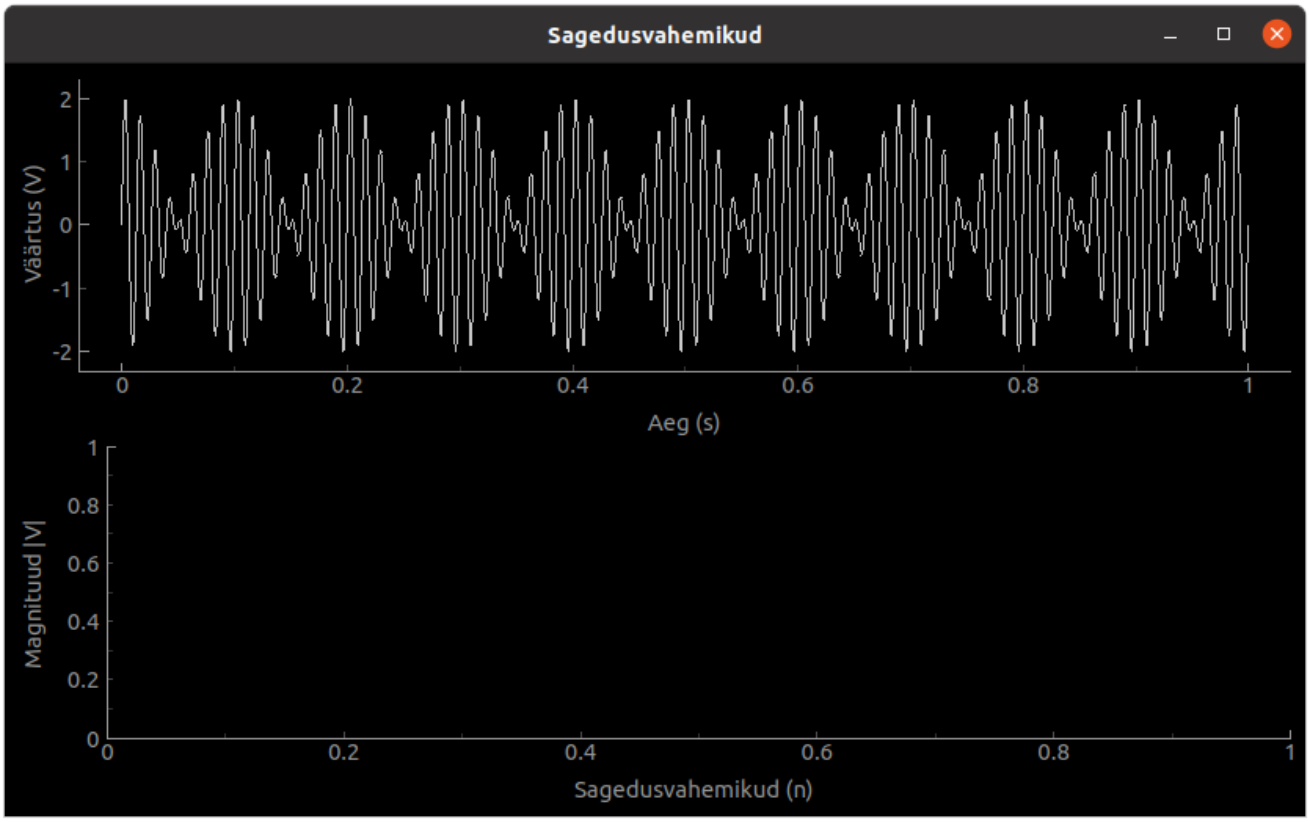
Sissejuhatus

Nüüdseks olete tutvunud diskreetse Fourier' pöördega ja selle pöördteisendusega. Loodetavasti olete aru saanud, kuidas sagedusruumi liikuda ja sealt aja domeeni naasta. Järgnevas ülesandes õpite loodud sagedusruumi graafikuid tõlgendama ning uurite mõningaid DFT-ga seotud piiranguid.

Ülesande kirjeldus

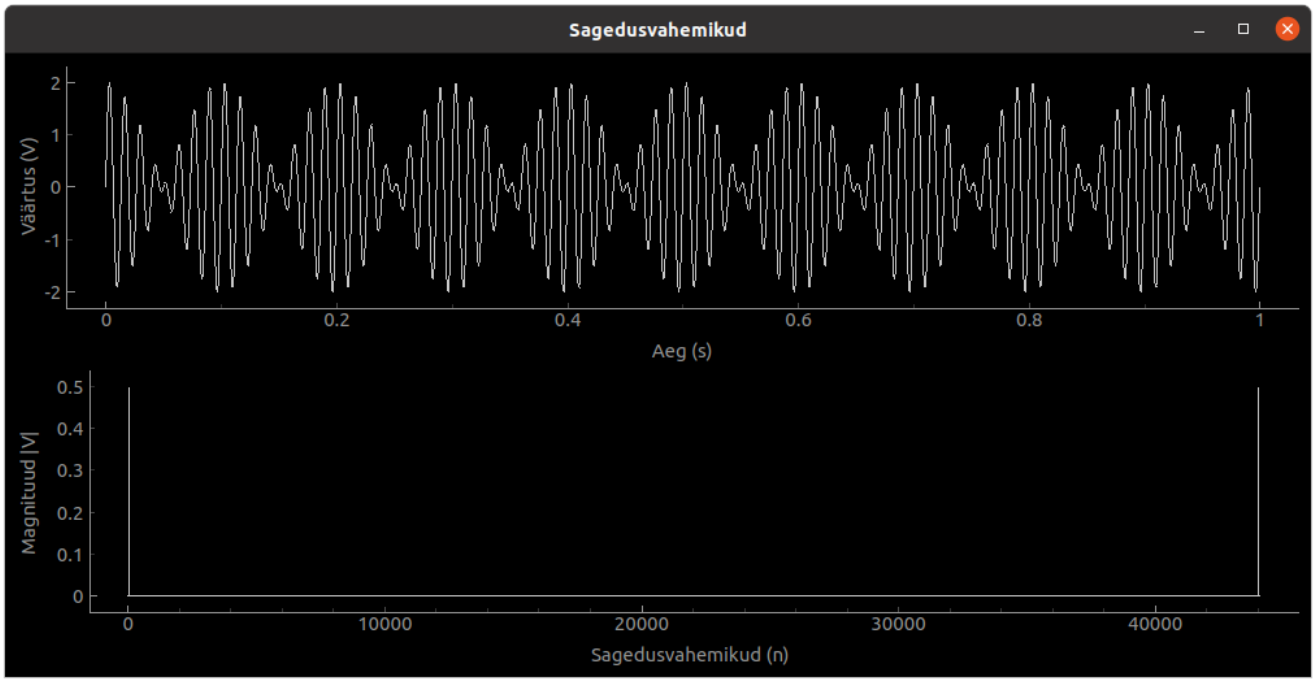
1) Loome uuritava signaali

Esmalt looge kaks siinust sagedustega 70 Hz ja 80 Hz amplituudiga 1V, mida on ühe sekundi jooksul sämplitud kiirusega 44,1 kHz. Liitke need kokku üheks signaaliks ning kujutage seda graafikul ajas koos korrektsete x,y-telje väärtustega.



Nüüd rakendage oma sisendsignaalile Fourier' numpy teegi kiirteisendust. Me ei kasuta siin 2. ülesandes loodud FFT arvutamise funktsiooni, kuna käesolevas ülesandes ei ole meie sisendsignaali pikkus 2 aste.

Pange tähele, et see funktsioon arvutab kompleksarvulise DFT. Seetõttu on arvutatud DFT tulemuste magnituud avaldatav kui $|X[k]| = \sqrt{Re(X)^2 + Im(X)^2}$



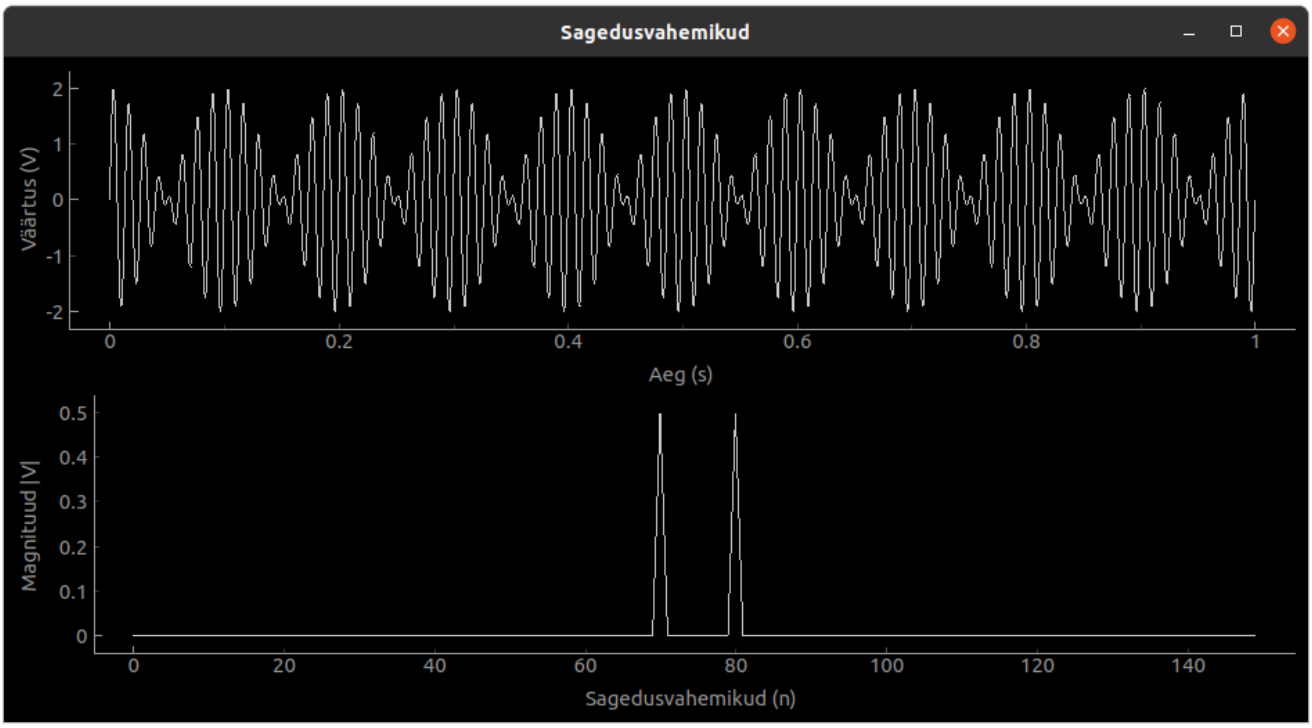
Te saate 44100 punktilise DFT, kuid esmapilgul võib tunduda, et FFT algoritm tuvastas ainult kaks sagedust, ühe graafiku alguses ja teise graafiku lõpus. Suurendage hiirt kasutades kumbagi nurka. Siis peaks nägema hoopis kahte tulemust positsioonidel 70, 80 või 44020, 44030. Tähelepanelikumad teist on juba kindlasti märganud, et esimesed kaks positsiooni langevad täpselt teie kahe sageduse peale. Teised kaks positsiooni on sämplimise sagedus, millest on lahutatud kummagi signaali sagedus. Millest selline kokkusattumus? Meenutage 4. praktikumi.

DFT lahutab `N` pikkusega sisendsignaali `N` erinevasse sagedusvahemikku. Seetõttu on DFT lahutusvõime (Δ_{DFT}), ehk võime eristada kahte erinevat üksteisele lähedal oleva sagedusega signaali, proportsionaalne sisendsignaali pikkusega. Mingis mõttes tundub see loogiline, et mida kauem te sisendsignaali "kuulate", seda rohkem informatsiooni te selle kohta teada saate. Järelikult suudate eristada rohkemate signaalide vahel ning jaotada sisend sagedusruumis rohkemate sagedusvahemike vahel. Matemaatiliselt on see väljendatud järgnevalt:

$$\Delta_{DFT} = \frac{\text{sämplimissagedus (Hz)}}{\text{signaali pikkus (N)}} \qquad (9)$$

Praegusel juhul on meie DFT lahutusvõime $\frac{44100Hz}{44100}$ ehk siis 1 Hz jaotuse kohta. Tipp indeksil 70 on siis 70 Hz, tipp indeksil 80 on siis 80 Hz ning tipud asukohtadel 44020, 40030 on vastavalt 44020 Hz ja 44030 Hz. Kuid siinkohal tuleb meenutada Nyquist-Shannon teoreemi ning mõelda, kas me saame usaldada FFT poolt arvutatud sagedusi üle 22050 Hz piiri? Lihtne vastus on, et ei saa. Paarisarvulise sisendipikkusega FFT tagastatud tulemused üle $\frac{N}{2}+1$ on aliased, paarituarvulise pikkuse korral on see $\frac{N}{2}$. Aliaste kohta võite kuulda ka terminit negatiivsed sagedused. Enamasti need sagedused huvi ei paku ja neid võib ignoreerida.

Järelikult võite FFT tulemustest võtta esimesed $\frac{N}{2}$ või $\frac{N}{2}+1$ tulemust kuna ülejäänud on FFT poolt arvutatud aliased. Samuti, kuna teate, et meie ideaaliseeritud signaalis on ainult kaks madalat sagedust, siis võtke vaatluse alla näiteks esimesed 150 DFT väärtust. Teie graafik peaks nüüd välja nägema järgnev:



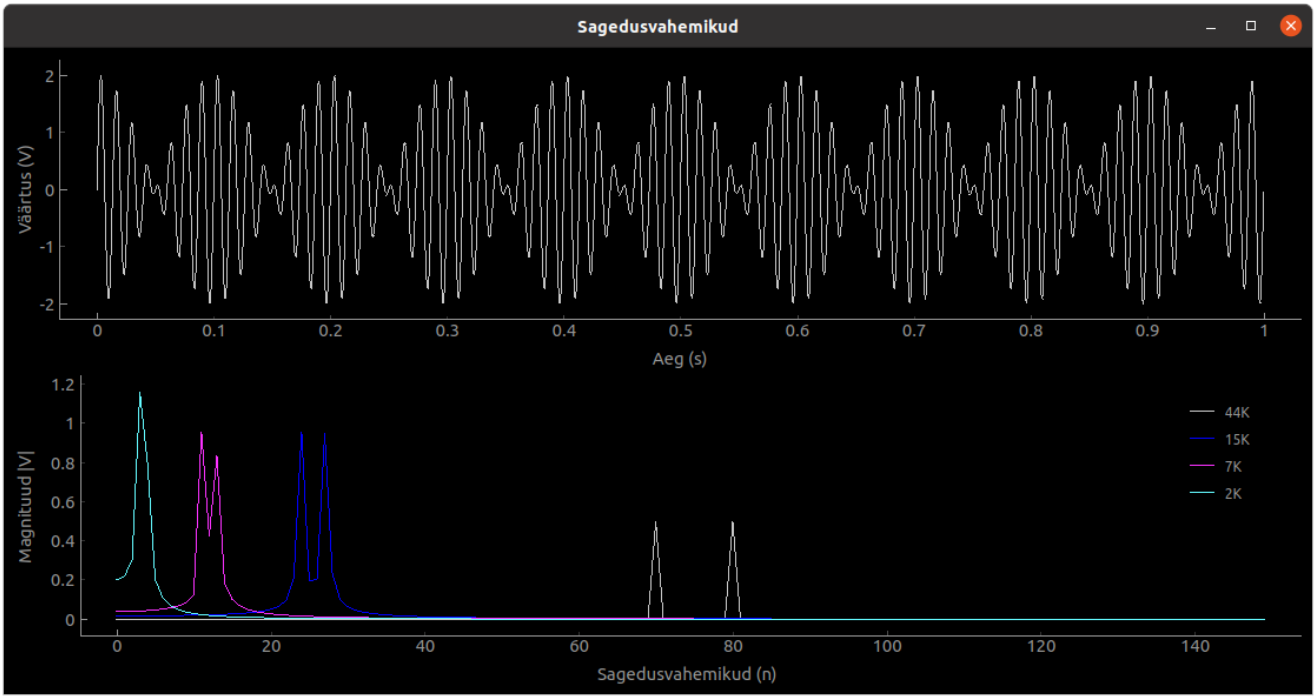
Nüüd võib tekkida küsimus y-telje ühikute kohta ning kõige lihtsam vastus sellele on, et need on täpselt samad nagu ajasignaalil (V) ning teid eriti ei huvita. Teid huvitab, kui tugevad on erinevad sagedused üksteise suhtes ning selle jaoks pole vaja ühikuid. Lisainfot leiab selle kohta veel siit.

2) Uurime DFT lahutusvõimet

Võtke algselt genereeritud signaal ning selle DFT kasutades taaskord `numpy.fft.fft` funktsionaalsust. Võtke algsest signaalist järgnevad jupid ning kujutage nende sageduskomponente ühel graafikul:

1. 44 100 punktiline lõik
2. 15 000 punktiline lõik
3. 7000 punktiline lõik
4. 2000 punktiline lõik

Lõpptulemus peaks välja nägema järgnev:



Graafiku loetavuse huvides on mõistlik FFT väljund läbi jagada sisendi pikkusega.

Lisaks leidke vastused järgnevatele küsimustele:

1. Kuidas saada graafikul olevast x-telje sagedusvahemiku indeksist vastav signaali sagedus? Teisisõnu, leidke valem, mis suudab öelda, mistahes tipu puhul graafikul, mitme hertsise sagedusega on tegu. Teid aitab Valem 9.
2. Arvutage iga ülaltoodud signaali pikkuse puhul tuvastatud sagedused.
    a. Kui palju erinevad need tegelikest sagedusdest ning miks?
3. Katsetage, mis on väikseim sämplite arv, millega suudate eristada kahte tippu.
    a. Leidke vastavad sagedusvahemikud. Lugege need välja graafikult.
    b. Demonstreerige leitut juhendajale.
4. Kui me ei tea sämplimise kiirust, kas saame öelda, mis sagedused on signaalis?

Näidake oma lahendust praktikumi juhendajale ning tehke commit!

Kolmas kodutöö

1. Valem 9 näitab meile, et mida pikem signaal, seda täpsemalt suudab DFT sagedusi eristada. Seega püstitame ühe hüpoteesi - "Võtame algsest signaalist 2000, 7000 ja 15000 sämplit ning lisame neile nulle kuni iga järjend hoiab endas 44100 sämplit. Seega meie DFT lahutusvõime on nüüd 1 Hz ning me näeme kahte sageduskomponenti ka 2000 sämpli pikkuse signaali puhul!".

   Testige juhendajate hüpoteesi oma programmiga.

   Eeldatav vastus:

   - toetub praktikumis loodud programmi graafilisele väljundile;
   - sisaldab ühe lõigu jagu arutlevat teksti, mis põhjendab tulemust.

3. Kodutöö ülesanded

Vastuste esitamine toimub Moodle keskkonnas. Kui kasutate väliseid allikaid, siis tuleb neile ka viidata!

1. ülesanne

1. Koostage skeem enda teises ülesandes loodud FFT funktsiooni rekursiooni läbimisest. Sisendina kasutage signaali, milles on 8 punkti. Skeemi võib joonistada paberile ning tuleks sellisel juhul lisada fotona kodutöö faili ning viidata otselingiga täissuuruses failile, mis asub teie repositooriumis.

   Eeldatav vastus:

   - skitseerib skeemi / puu, kus on kujutatud andmepunktitde töötlemise järjekorda FFT algoritmis;
   - selgitab ühe kuni kahe lõiguga skeemi / puu ülesehitust toetudes enda koodile.

2. Mõõtke oma implementeeritud FFT ja P05 implementeeritud DFT algoritmi kiirust erinevate sisendite korral ning kujutage tulemusi graafikul.

   Eeldatav vastus:

   - sisaldab vähemalt viie erineva sisendiga sooritatud mõõtmiste tulemusi, kusjuures vähima ja suurima sisendi pikkuste vahe peab olema vähemalt tuhat korda;
   - võrdleb graafikul ning kirjeldab paari lausega, kui lähedal on teie implementatsioon esimeses praktikumiülesandes toodud keerukusele $O(n\log(n))$.

2. ülesanne

1. Miks ei mõjutanud `overlap_add` funktsioonis nullide lisamine kuuldavat väljundit? Nulle lisati nii impulsskostele kui ka konvoleeritavale signaalile.

   Eeldatav vastus:

   - arutleb ühe lõigu piires antud teemal;
   - kasutab varasematest praktikumidest saadud signaalide kombineerimise alaseid teadmisi;
   - toob vajadusel näite, millega oma selgitust toetada (ei ole kohustuslik).

3. ülesanne

1. Valem 9 näitab meile, et mida pikem signaal, seda täpsemalt suudab DFT sagedusi eristada. Seega püstitame ühe hüpoteesi - "Võtame algsest signaalist 2000, 7000 ja 15000 sämplit ning lisame neile nulle kuni iga järjend hoiab endas 44100 sämplit. Seega meie DFT lahutusvõime on nüüd 1 Hz ning me näeme kahte sageduskomponenti ka 2000 sämpli pikkuse signaali puhul!".

   Testige juhendajate hüpoteesi oma programmiga.

   Eeldatav vastus:

   - toetub praktikumis loodud programmi graafilisele väljundile;
   - sisaldab ühe lõigu jagu arutlevat teksti, mis põhjendab tulemust.