Fourier

Sisukord

Aastal 2021 on vastutav juhendaja: Meelis Pihlap <pihlap@ut.ee> 2021-04-06 | 12:32:49 +0300

1. Ülevaade

Juhendist on olemas ka PDF versioon.

Tänases praktikumis vaatame signaale nii aja- kui ka sagedusruumis. Praktikumi lõpuks oskate liigutada signaali ajadomeeni ja sagedusruumi vahel ning kujutada oma tulemusi graafiliselt.

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. PyQt (5.1)
3. NumPy (1.20)

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

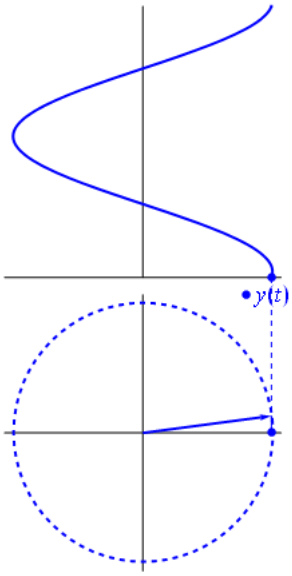
Repositooriumisse on lisatud uut sisu. Selle sisu kasutamiseks peate ta oma harusse liitma.

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

Sissejuhatus

Eelmises praktikumis lähendasime diskreetseid, perioodilisi signaale sinusoidide summana. Meenutame, et sinusoidi kirjeldavad sagedus, amplituud, faasinihe ja vertikaalnihe. Visuaalselt on hea seda informatsiooni esitada kasutades ühikringi ja polaarkoordinaate. Tuletamaks meelde, kuidas on seotud omavahel ringid ja sinusoidid, lugege läbi kaks peatükki juba tuttavast veebimaterjalist (https://jackschaedler.github.io/circles-sines-signals/sincos.html).



(https://upload.wikimedia.org/wikipedia/commons/8/89/)

Pildil olevat fikseeritud pikkusega vektorit, mille tipp liigub mööda ühikringi, nimetatakse faasoriks.

- Sagedus väljendab kui kiiresti teeb faasor täispöörde.
- Amplituud määrab faasori pikkuse.
- Faasinihe määrab faasori nurga algväärtuse.

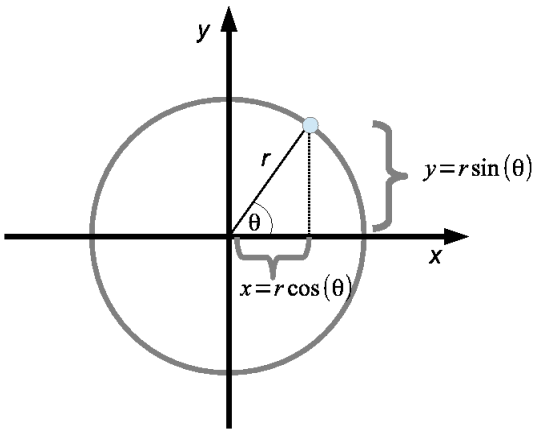
1. Ülesanne: Faasori visualiseerimine

Selle ülesande raames implementeerime ühe faasori liikumise. Ülesande lihtsustamiseks on antud aluskood `yl1_phasor_sum.py`

1. Märgake koodis rida `Phasor = namedtuple('Phasor', 'frequency magnitude phase') # Ühe faasori omadused`
   a. Siin on kirjeldatud faasor Pythoni namedtuple (https://docs.python.org/3/library/collections.html#collections.namedtuple) objektina. Tutvuge dokumentatsiooniga, et õppida seda kasutama.
2. Faasori joonistamiseks implementeerige `Main` klassis `update` funktsiooni sisu ja klassis `PhasorSumVisualizer` funktsioonide `calculatePhasors()` ja `paint()` sisu.
   a. Täpsed nõuded funktsioonide sisule on esitatud kommentaaridena vastavates funktsioonides.
3. Leidke aluskoodist muutuja, mis määrab faasori animatsiooni sammu.

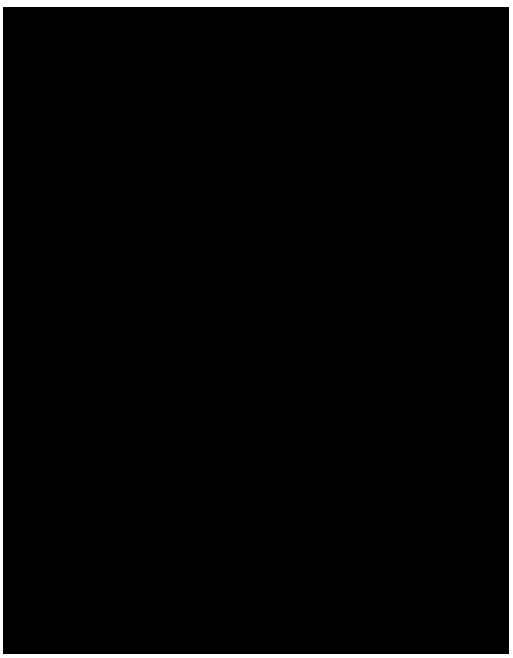
Põhiprogramm on jaotatud kahe klassi vahel: `Main` ja `PhasorSumVisualizer`. `PhasorSumVisualizer` arvutab igal ajahetkel `n` faasori koordinaadid ja joonistab selle ekraanile. `Main` uuendab `PhasorSumVisualizer` objekti igal visualiseerimise sammul ning uuendab ajaväärtusi.

Faasori koordinaatide arvutamisel on abiks järgnev joonis.



(http://www.thephysicsmill.com/2013/09/23/between-being-and-non-being-imaginary-numbers/)

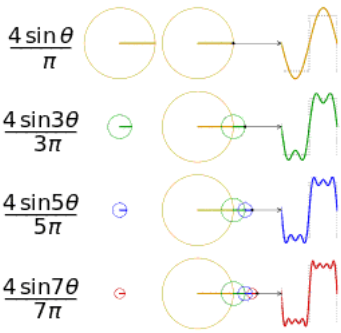
Näide oodatud tulemusest on esitatud allpool olevas videos.



0:00 / 0:01

2. Ülesanne: Ruutsignaali lähendamine

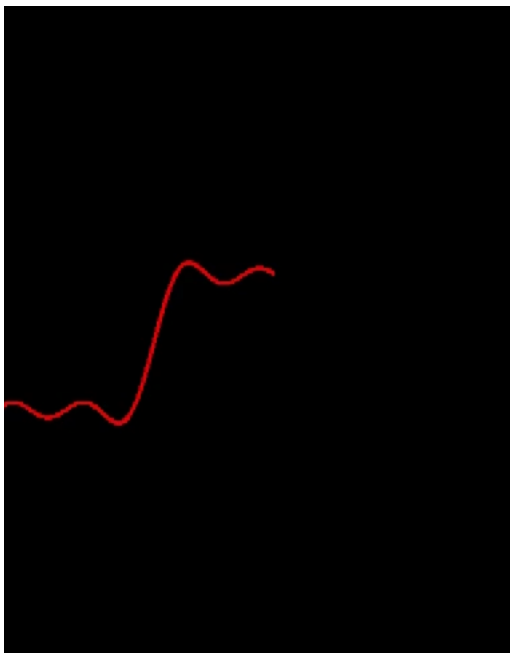
Faasoreid liites on võimalik esitada mistahes perioodilist signaali. Näiteks ruutsignaali puhul näeb sobivaid faasoreid liites sellist väljundit:



(https://commons.wikimedia.org/w/index.php?curid=32504389)

Vaadates pildil näidatud valemeid näete, et suurimale kollasele faasorile vastab valem $\frac{4\sin(\theta)}{\pi}$. Selle all olev valem $\frac{4\sin(3\theta)}{3\pi}$ tähistab väiksemat rohelist ringi. Selle valemiga saab arvutada vastava faasori tipu asukoha y-teljel ajahetkel θ. Iga ring tähistab ühte faasorit ning faasorite summeeritud mõju näitab iga valemi kõrval olev graafik. Mida rohkem erinevaid faasoreid välja arvutame ja kokku liidame, seda täpsemalt suudame kujutada ruutsignaali.

Teie eesmärgiks on eelneva info põhjal kirjutada programm, mis arvutaks ruutsignaali faasorite summana ja visualiseeriks tulemust. Näide oodatud tulemusest on esitatud allpool olevas videos.



Esimene etapp - Mitu faasorit

1. Funktsioon `square_wave()` võtab sisendiks muutuja `term`, mis tähistab, mitmendat faasorit praegusel ajahetkel soovitakse arvutada. Funktsioon peab tagastama seda faasorit esitava namedtuple (https://docs.python.org/3/library/collections.html#collections.namedtuple) tüüpi objekti, mille väljadeks on vastava faasori sagedus ja magnituud.
   a. Pange tähele, et see funktsioon arvutab ainult faasori amplituudi ja sageduse, mitte lõppkoordinaadid.
2. Klassi `Main` meetodis `keyPressEvent`, loetakse klaviatuurilt nooleklahvi sisendeid ▲ või ▼ ja vastavalt sellele suurendatakse või vähendatakse muutujat `self.nr_of_terms` ühe võrra iga klahvivajutuse kohta. See muutuja kontrollib, mitme faasori panust lõpptulemusse lisatakse. Faasorite lisamine tuleb implementeerida `Main` klassi `update` meetodis.
3. Vaadates esialgset pilti näete, kuidas iga ringi raadiuse tipus asetseb järgmine ring/faasor.
   a. Klassi `PhasorSumVisualizer` meetodis `calculatePhasors` tuleb leida `self.phasors` järjendi põhjal iga faasori keskpunkt ja raadius ning salvestada need vastavatesse järjenditesse.
   b. Samuti summeerime kokku iga faasori poolt avalduva `x,y` "nihke" mõju ning `calculatePhasors` funktsiooni lõpuks on muutujates `self.x_pos` ja `self.y_pos` lõplik `x,y` positsioon, milles on võetud arvesse kõikide arvutatud faasorite mõju.
4. Viimaks on vaja implementeerida `PhasorSumVisualizer` klassi meetod `paint()`.
   a. Siin tuleb joonistada `self.circles` järjendis oleva info põhjal kõikide faasorite ringid ja vektorid. Abiks on PyQt Dokumentatsioon (https://doc.qt.io/qt-5/qpainter.html)

Teine etapp - Projektsioon

Eesmärk on kujutada joonisele faasorite summa y-telje projektsioon. Selle jaoks on osa eeltööd juba tehtud, teil tuleb oma eelmises etapis saadud tulemused siin visualiseerida.

5. Täiendage klassi `Main` meetodit `update()`.
   a. Lisage viimane faasorite abil välja arvutatud punkt joonisele kasutades funktsiooni `add_point()`. Kuna tegeleme y-telje projekteerimisega, siis x-koordinaadi väärtuseks võib esialgu panna 0.
6. Funktsiooni `drawPath()` sisemine loogika nihutab punkte automaatselt edasi, selleks muutke välja `self.X_POS_INCREMENT` väärtus 0.25'ks.
   a. Välja `self.WAVE_MAX_LENGTH` väärtus määrab, mitut punkti lõppsignaalist kuvatakse.
   b. Käivitage programm. Üles noolt vajutades peaks nüüd olema näha, et tavaline siinus muutub järjest enam ruutsignaali kujuliseks, mida enam erinevate sageduste mõju omavahel liita. Kui seda ei juhtu, on tõenäoliselt viga funktsiooni `square_wave()` implementatsioonis.
7. Praegune joonise paigutus ei ole optimaalne. Selle parandamiseks asendage funktsiooni `add_point()` väljakutses signaalipunkti x-koordinaadi asukohta määrav muutuja muutujaga `self.X_POS_OFFSET`
   a. Sobiv `X_POS_OFFSET` väärtus valige ise.
8. Ülal toodud näites on muutuja `self.nr_of_terms` väärtus 3.

Faasoreid ja lõppsignaali ühendav joon tuleb joonistada klassi `Main` meetodis `paintEvent()`

Näidake oma ruutsignaali loomise lahendust praktikumi juhendajale ning tehke commit!

---

3. Ülesanne: Diskreetne Fourier' pööre

Sissejuhatus

Eelmises ülesandes kujutatud signaalid on levinud ja tuntud matemaatiliste esitustega. Kuid kuidas esitada signaale, mille matemaatiline kuju ei ole teada? Siinkohal tuleb abi saamiseks pöörduda Fourier' teisenduse poole. Oma olemuselt lahutab Fourier' pööre sisendsignaali erinevateks sageduskomponentideks ning igal liikmel on oma sagedus ja amplituud, mis määrab, kui tugevalt on see liige esindatud esialgses signaalis. Internetis on väga palju informatsiooni Fourier' teisenduse kohta. Alustuseks soovitame lugeda seda õpikut (https://www.dspguide.com/ch8.htm), seda artiklit (https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/) ning vaadata järgnevat videot (https://www.youtube.com/watch?v=spUNpyF58BY).

Ülesande kirjeldus

Järgnev ülesanne koosneb neljast etapist:

1. sisendsignaali kujutamine ajas,
2. diskreetse Fourier' teisenduse (DFT) implementeerimine,
3. signaali kujutamine sagedusruumis,
4. DFT pöördteisendus.

Esimene etapp

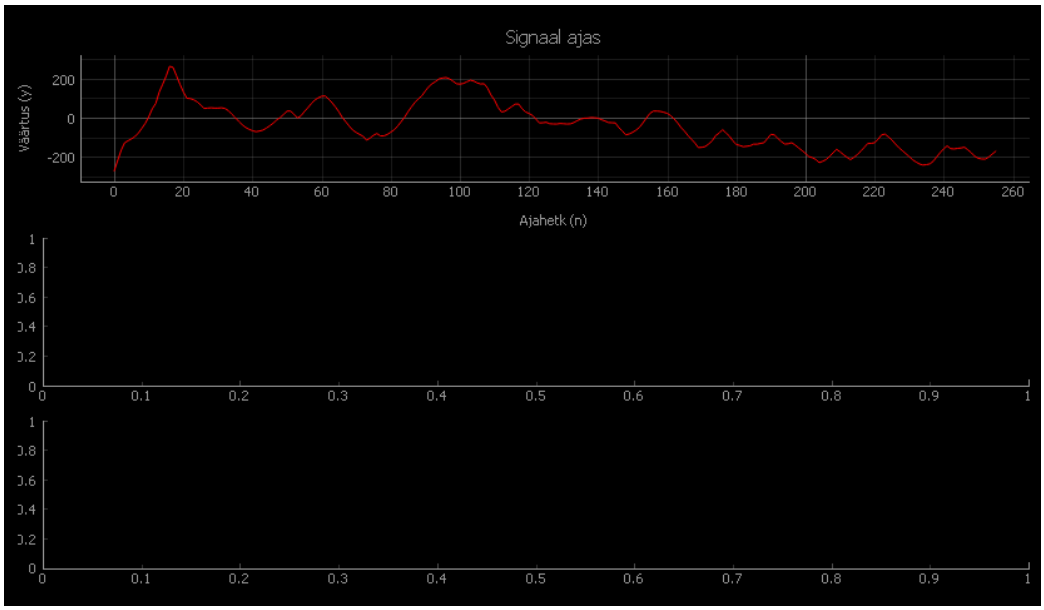
Selle etapi tulemusena valmib graafiku põhi, millel on kujutatud sisendsignaali ning on loodud kohad sagedusruumi ja kaks pööret läbinud signaali kujutamiseks.

Oma repositooriumist leiate te faili `sample_signals.py` , mis sisaldab nelja juhendajate pool genereeritud signaali. Selles ülesandes on teil vaja ainult kahte esimest signaali. Alustades soovitame graafiliselt kujutada esimest signaali, ning seejärel teha sama ka teise signaaliga.

Pythonis on võimalik ühes failis esinev muutuja teise faili nimeruumi lisada kasutades järgmist süntaksit.

```python
from sample_signals import signal1
```

Looge PyQtGraphi abil järgnev graafik.



Teine etapp

Nüüd, kus olete näinud mõlemat signaali ajas, on aeg viia need signaalid sagedusruumi, alguses üks ja siis teine. Fourier' teisendusi on erinevaid, sõltuvalt sisendandmete kujust ja perioodilisusest, kuid arvutid suudavad töötada ainult informatsiooniga, mis on diskreetne ja lõplik. Seega on kõige levinum Fourier' teisendus diskreetne Fourier' teisendus, ehk DFT. Rohkem infot erinevate Fourier' teisenduste kohta leiab siit (https://www.dspguide.com/ch8/1.htm).

Teie ülesandeks on implementeerida diskreetne Fourier' teisendus, mis võtab sisendiks mistahes signaali ajas (1D järjend) ning tagastab järjendi vastavate sageduskomponentidega. Graafiliselt näeb selle funktsiooni töö välja järgnevalt, kus punane tähistab signaali ajas, siniselt on tähistatud sama signaal sagedusruumis.

(https://commons.wikimedia.org/w/index.php?curid=24830373)

Diskreetse Fourier' teisenduse (https://et.wikipedia.org/wiki/Diskreetne_Fourier%27_teisendus) valem on järgnev:

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n] \cdot [cos(\frac{2\pi kn}{N}) - jsin(\frac{2\pi kn}{N})] \quad (1)$$

$X[k]$ on sageduskomponendi k kogus terves signaalis.

k on praegusel hetkel töödeldav sagedus (0 kuni N-1).

N on uuritava signaali punktide arv.

n on praegu töödeldava ajapunkti number (0 kuni N-1).

$x[n]$ on signaali väärtus ajahetkel n.

j tähistab siin ja edaspidistes valemites imaginaarühikut.

Internetis leidub mitmeid erinevaid kujusid (https://math.stackexchange.com/questions/58163/dft-why-are-the-definitions-for-inverse-and-forward-commonly-switched) Fourier' pöörde valemile. Need kõik on õiged seni, kuni te neid segamini ei kasuta.

Valemis on korrutatud siinuskomponenti imaginaarühikuga. Pythonis on kompleksarvude tugi, samas mitmetes teistes programmeerimiskeeltes selline tugi puudub. Seetõttu kujutame universaalsema lahenduse nimel kompleksarvud komplekstasandi (https://et.wikipedia.org/wiki/Komplekstasand) punktidena. Selleks kasutame ristkoordinaatide süsteemi, kus abstsisstelge nimetatakse reaalteljeks, ordinaattelge imaginaarteljeks. Tasandi igal punktil on kaks reaalarvulist koordinaati ning selles koordinaatide süsteemis saame kasutada DFT trigonomeetrilist vormi. Rohkem infot leiab siin lingitud loengumaterjalist (http://www.staff.ttu.ee/~mvaljas/3710/loeng_02.pdf).

Looge funktsioon `dft_trig`, mis võtab sisendiks 1D signaali ning väljastab järjendi pikkusega N sageduskomponenti. Sageduskomponentide loomiseks kasuta taaskord namedtuple (https://docs.python.org/3/library/collections.html#collections.namedtuple) funktsionaalsust. Iga sageduskomponent peab omakorda sisaldama:

1. sageduskomponendi sagedust ehk selle indeksit DFT pöörde tsüklis;
2. reaalosa;
3. imaginaarosa;
4. sageduskomponendi suurus (https://et.wikipedia.org/wiki/Suurus) (inglise k magnitude (https://en.wikipedia.org/wiki/Magnitude_(mathematics)));
5. faasi (https://en.wikipedia.org/wiki/Atan2) radiaanides.

Kolmas etapp

Kui teid huvitab juba praegu genereeritud graafiku x- ja y-telje korrektne tõlgendamine, siis soovitame vaadata seda videot (https://youtu.be/FjmwwDHT98c?t=1099). Sagedusruumi täpsem uurimine on järgmise praktikumi eesmärk.

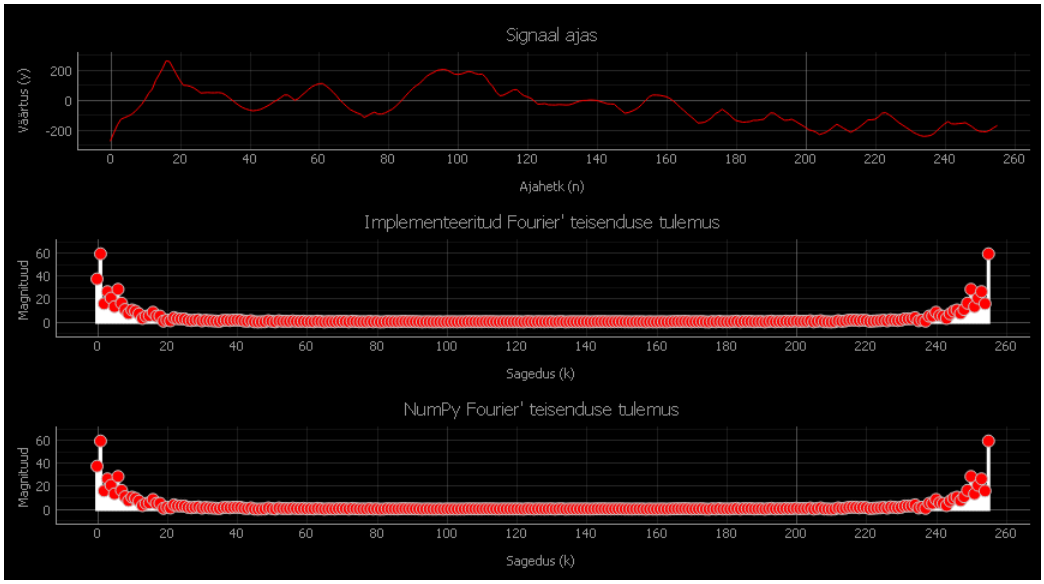
Siin etapis tegelete leitud DFT sagedusruumi graafikul kujutamisega. Selleks, et oma loodud funktsiooni kontrollida, saate kasutada numpy.fft.fft (https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html#numpy.fft.fft) funktsionaalsust.

Oma funktsiooni kontrollimiseks tehke järgnevat:

1. Võtke igast arvutatud sageduskomponendist selle magnituud ning lisage need uude järjendisse.
2. Kujutage saadud tulemust keskmisel graafikul.
3. Kasutage NumPy FFT (https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html#numpy.fft.fft) funktsiooni, et arvutada esialgse signaali Fourier' teisendus.

a. Kuna NumPy kasutab enda sisemuses natukene teistsugust Fourier' pöörde valemit, tuleb saadud tulemusest võtta absoluutväärtus (magnituud) ja jagada terve tulemus signaali pikkusega.

4. Keskmine ja kõige alumine graafik peaksid välja nägema identsed. Kui see ei ole nii, siis oleme päris kindlad, et teie implementatsioonis on kusagil viga.



Neljas etapp

Nüüdseks olete oma signaaliga edukalt sagedusruumi jõudnud, kuid elus võib ette tulla hetki, kus tahate ka sagedusruumist tagasi ajadomeeni astuda. Õnneks on selleks otstarbeks Fourier' pöördteisenduse (https://www.dspguide.com/CH31.PDF) valem, mis on toodud allpool.

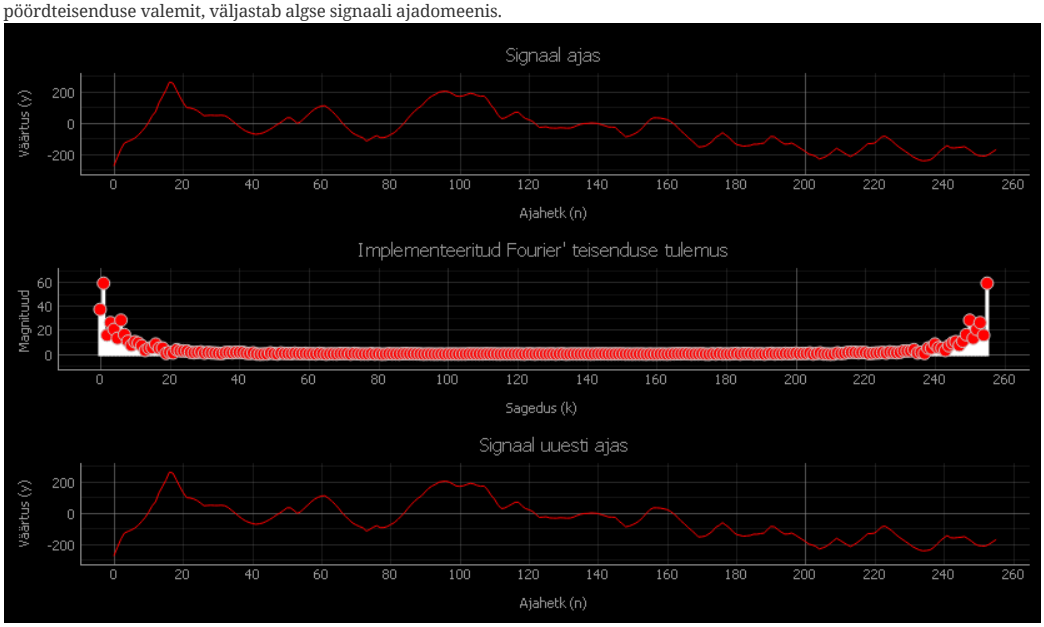
$$x[n] = \sum_{k=0}^{N-1} X[k] \cdot e^{\frac{j2\pi kn}{N}} = \sum_{k=0}^{N-1} \left[Re(X[k]) \cdot \left(cos(\frac{2\pi kn}{N}) + jsin(\frac{2\pi kn}{N}) \right) \right] - \sum_{k=0}^{N-1} \left[Im(X[k]) \cdot \left(sin(\frac{2\pi kn}{N}) - jcos(\frac{2\pi kn}{N}) \right) \right] \quad (2)$$

$Re(X[k])$ tähistab kompleksarvu reaalosa.

$Im(X[k])$ tähistab kompleksarvu imaginaarosa.

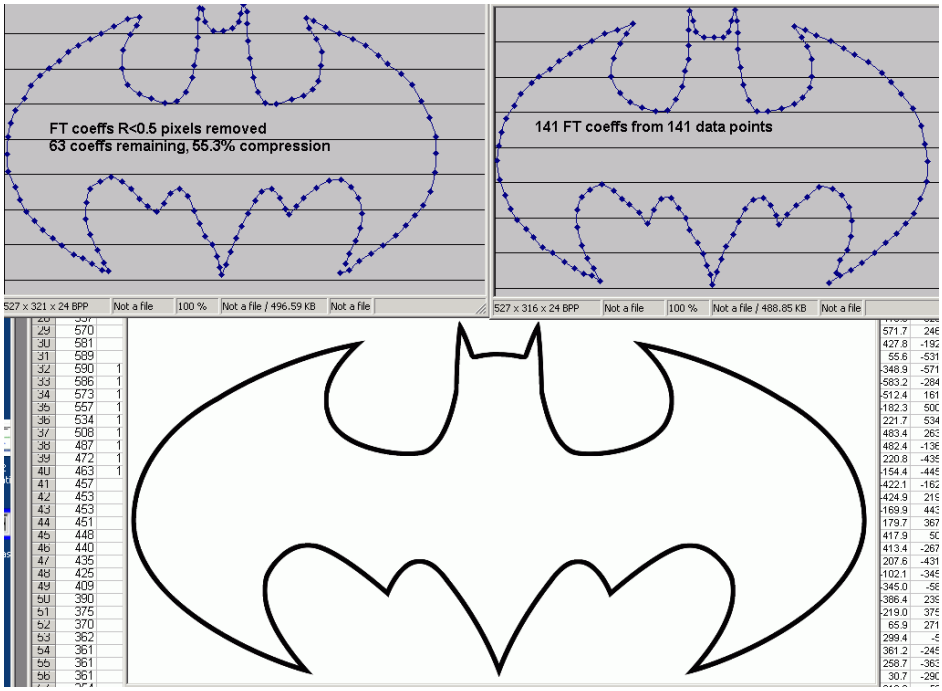
Ülejäänud notatsioon on sama, mis eelnevate valemite puhul. Siit esitusest on ka hea näha, miks matemaatikud eelistavad antud valemi väljakirjutusele Fourier' pöördteisenduse valemi komplekskuju.

Looge funktsioon `idft_trig`, mis võtab sisendiks eelnevalt loodud sageduskomponendid ning kasutades Fourier' pöördteisenduse valemit, väljastab algse signaali ajadomeenis.



Näidake oma lahendust praktikumi juhendajale ning tehke commit!

4. Ülesanne: Diskreetne Fourier' teisendus kahemõõtmeliste signaalidega



Mistahes pilti võib tõlgendada kui perioodilist 2D signaali, mis koosneb x, y koordinaatide paaridest. Selles ülesandes kasutame eelpool loodud funktsionaalsust, et joonistada mistahes perioodilist 2D signaali, kasutades Fourier' teisenduse algoritmi.

Ülesande kirjeldus

Selle ülesande osana kasutame esimese ülesande lahendust.

Jaotame ülesande taas etappideks.

1. Genereerime testimiseks kahemõõtmelise signaali, mis moodustab ringi
2. Leiame sageduskomponendid mõlema dimensiooni jaoks kasutades DFT-d
3. Visualiseerime sageduskomponendid eraldi telgedel
4. Leiame funktsiooni väljundi kombineerides kaks telge

Testsignaali genereerimine

1. Looge testimiseks funktsioon `create_circular_signal` , mis võtab valikuliseks sisendiks muutuja `scaler` ning tagastab kaks 100-elemendilist järjendit loodud ringi x ja y väärtustest.
   a. 100 elemendilised järjendid peavad olema x, y koordinaadid nurkade vahemikust $[0, 2\pi)$.
   b. Järjendite tagastamisel korrutatakse need läbi arvuga `scaler` . Kui funktsioon kutsutakse välja ilma sisendita, on `scaler` muutuja vaikimisi väärtus 50.

Sageduskomponentide leidmine

1. Võtke kummastki tagastatud järjendist `Main` klassi *`init`*`()` funktsioonis Fourier' pööre ning salvestage tulemused vastavalt `self.fourier_x` ja `self.fourier_y` muutujatesse.
   a. Ilusama vaatepildi jaoks sorteerige saadud järjendid suuruse järgi kahanevalt `sorted(järjend, key=lambda x: x.mag, reverse=True)`

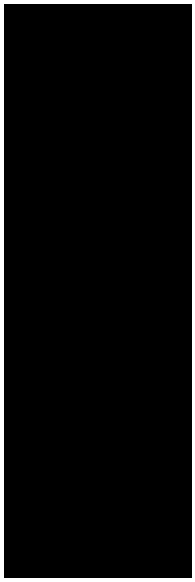
Faasorite summeerimine

1. Kuna tahame, et üks faasori summa oleks teisega 90 kraadi nihkes ja et faasoritel saaks olla faasinihe, siis peame oma faasorite visualiseerijat veidi täiustama.
   a. Liitke klassi `PhasorSumVisualizer` meetodis `calculatePhasors()` koordinaatide arvutamisel juurde sageduskomponendi faasinihe ja välja self.ROTATION_OFFSET väärtus.
2. Määrake klassi `Main` meetodis `init()` x_psv algkoordinaatideks (540, 100) ja y_psv algkoordinaatideks (150, 360).
3. Määrake `y_psv` välja `ROTATION_OFFSET` väärtuseks `-np.pi/2`

Väljundsignaali arvutamine

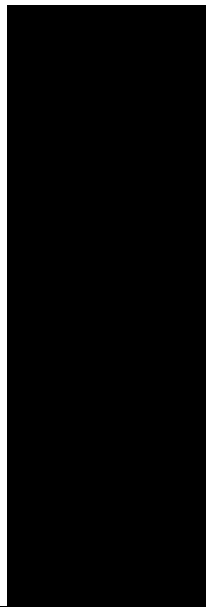
1. Klassi `Main` meetodis `update()` võtke mõlemast arvutatud punktist korrektsed x, y väärtused ning andke need argumendiks funktsioonile `self.add_point` .
2. Viimaks suurendage muutujat `self.current_angle` $\frac{2\pi}{\text{len}(\text{fourier})}$ võrra.

Lõpptulemus praegusel hetkel peab välja nägema järgnev:



0:00 / 0:05

Kasutades eelpool õpitut, täiendage oma programmi nii, et lõpptulemus näeks välja järgnev:



0:00 / 0:04

Arvestuse saamiseks tuleb demonstreerida oma programmi, kasutades mõlemat kujutist failist `sample_signals.py`.

Näidake oma lahendust praktikumi juhendajale ning tehke commit!

3. Lisaülesanded

Enne lisaülesannete lahendamist on kohustuslik ette näidata kõigi põhiülesannete töötavad lahendused.

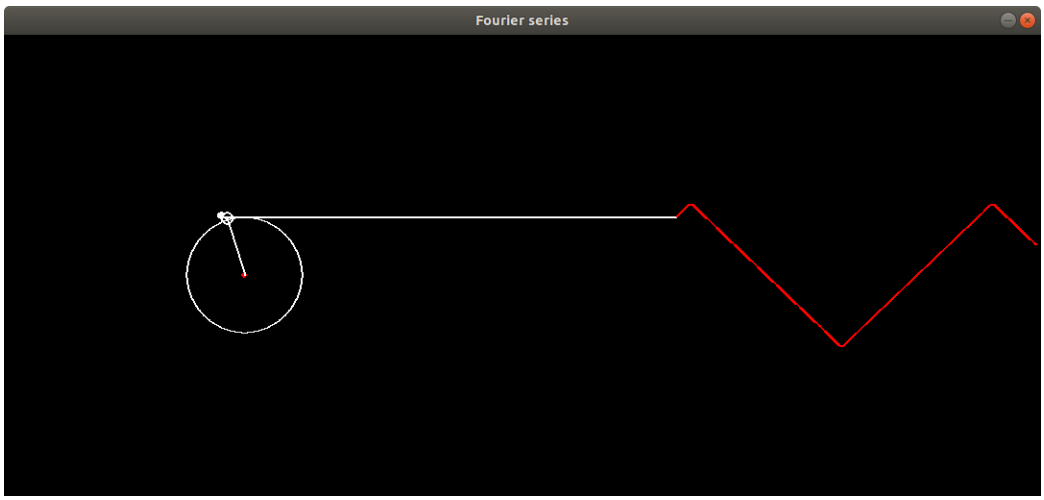
5. Ülesanne: Kolmnurga valemi implementeerimine (0.5 punkti)

Sissejuhatus

Esimeses ülesandes implementeerisime ruutsignaali ja saehambasignaali valemid. Selles lisaülesandes tuleb teil implementeerida kolmnurksignaali valem.

Ülesande kirjeldus

Kui kasutaja vajutab klahvi `3`, siis alustatakse kolmnurksignaali genereerimist. Kolmnurksignaali valemit leiate siit (http://bilimneguzellan.net/en/purrier-series-meow-and-making-images-speak/).



Näidake oma lahendust praktikumi juhendajale ning tehke commit!

6. Ülesanne: joonistuse DFT (2 punkti)

Sissejuhatus

Kuigi signaalide failist sisse lugemine ja nende graafiline kujutlemine on tore, saab viimast kohustuslikku ülesannet muuta veel huvitavamaks. Selle lisaülesande eesmärk on võtta sisendsignaaliks kasutaja poolt joonistatud kujund, seda töödelda ning, kasutades Fourier' pööret, graafiliselt taasesitada.

Ülesande kirjeldus

Programmi üldine tööloogika võiks välja näha järgnev:

1. Programm ootab, kuni kasutaja joonistab mingisuguse kujundi.
2. Salvestatud kujundist võetakse kompleksarvuline DFT.
3. Kasutades arvutatud sageduskomponente, taasesitatakse joonistatu ekraanile.
4. Kui kasutaja soovib uuesti joonistada, siis eelmine kujund unustatakse ning tööloogika alustab esimesest punktist.

Näidake oma lahendust praktikumi juhendajale ning tehke commit!

7. Ülesanne: Pilt signaaliks (2-4 punkti)

Sissejuhatus

Selle lisaülesande mõte on luua programm, mis võtab sisendiks mingisuguse pildi nagu näiteks sellised (https://duckduckgo.com/?q=colouring+sheet&iar=images&iax=images&ia=images). Programm genereerib nende põhjal järjendi (x, y) koordinaatidest nende läbimise järjekorras ning salvestab koordinaatpaarid faili, kus neid on võimalik importida 5. ülesande lahendusse.

Ülesande kirjeldus

Selles lisaülesandes on teil vabad käed kasutada mistahes metoodikat või teeke. Lahenduse demonstreerimiseks tuleb näidata koordinaatide genereerimist ja nende 5. ülesandesse importimist koos selle korrektse tööga. Lisaülesande eest antavate punktide arv sõltub teie esitatava lahenduse kompleksusest ja universaalsusest. Mida rohkem funktsionaalsust on implementeeritud ja mida töökindlam on lõpptulemus, seda rohkem punkte teile selle eest pakume.

4. Kodutöö ülesanded

Selle kodutöö eesmärk on ehitada tarkvaraline süntesaator, mis suudab mängida erineva tämbriga meloodiaid. Erinevalt eelmisest kodutööst määrab süntesaatori tämbri helifail, kus mängitakse mõnda instrumenti.

Teie loodud lahendus peab vastama järgnevatele nõutele:

1. Iga koodirida kodutöö failis peab olema põhjalikult kommenteeritud.
   a. `import` käskude seletamiseks ei pea olema detailsed kommentaarid.
2. Kõik kasutatavad muutujate ja funktsioonide nimed peavad olema läbi mõeldud.
   a. Kui juhendajad peavad teie koodi parandama või selles juppe sisse ja välja kommenteerima, olete selle punkti vastu eksinud.
3. Repositooriumis oleva `.py` faili nimi peab vastama eelnevates kodutöödes kehtestatud vormile.
   a. p05_kodutoo_perenimi_v1.py
4. Juhendajate kõrvade säästmiseks mitte kasutada põhihelidena kõrgemaid sagedusi kui 2000 Hz. Loodud kood tuleb laadida oma repositooriumi .py failina ning esitada .txt failina Moodle keskkonda lingi alla "5. kodutöö esitamine".

1. ülesanne

1. Leia ja laadi alla mõni endale meeldiva pilli heliklipp siit leheküljelt (https://freewavesamples.com/).
   a. Failides on kaks helikanalit, kombineeri need üheks (stereo to mono). Seda saab teha NumPy massiive manipuleerides, kuid võib ka muid tööriistu (nt Audacity) kasutada.
2. Soorita failile programmaatiliselt Fourier' teisendus ja leia sageduskomponendid.
3. Kuva tulemus graafikul.
   a. Leia graafiku põhjal signaali põhisagedus (https://et.wikipedia.org/wiki/P%C3%B5hisagedus) ja kirjuta see kommentaarina koodifaili.

2. ülesanne

1. Tekita Fourier' teisenduse tulemuste põhjal sama tämbri, kuid erineva põhiheliga ühe-sekundiline heliklipp ja salvesta see faili yl2.wav.

3. ülesanne

1. Täienda oma eelmise nädala kodutöö lahendust nii, et viisijupp mängitaks ette kasutades helifailist loetud pilli tämbrit.
2. Salvesta tulemus faili yl3.wav

Viimati uuendatud 2021-04-06 12:32:49 +0300