Konvolutsioon

Sisukord

Aastal 2021 on vastutav juhendaja: Erik Amor <erik.amor@ut.ee> 2021-03-03|12:54:22 +0200

1. Ülevaade

Juhendist on olemas ka PDF versioon.

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.

2. Töövahendid

Iga teegi järel on toodud klassi arvutites oleva paki versioon. See ei tähenda, et peate oma arvutis täpselt sama versiooni kasutama, 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. OpenCV 4.5
6. Audacity 2.4

Virtuaalkeskkonna pakkide nimekirja failis requirements.txt on lisatud uusi pakke. Oma virtuaalkeskkonda vajalike pakkide lisamiseks toimige järgnevalt:

```
# Lülitage virtuaalkeskkond sisse
kasutaja@arvuti:~$ source dsp-env/bin/activate

# Paigaldage oma virtuaalkeskkonnas vajalikud teegid
(dsp-env) kasutaja@arvuti:~$ python3 -m pip install -r requirements.txt
```

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`

3. Kohustuslikud ülesanded

1. Ülesanne: 1D konvolutsioon erinevate signaalidega

Sissejuhatus

Konvolutsioon on matemaatiline operatsioon kahe funktsiooni `f[n]` ja `g[n]` vahel, mille tulemusena tekib uus funktsioon `y[n] = (f ** g)[n]`. Saadud funktsioon kujutab tulemust, kus ühte funktsiooni on teise poolt mõjutatud. Konvolutsiooni saab teostada nii pideva kui diskreetse signaali korral. Selles aines vaatame digitaliseeritud (diskreetsete) signaalide konvolutsiooni.

Diskreetses ajadomeenis on konvolutsioon matemaatiliselt defineeritud järgnevalt:

```
\begin{equation}
    f[n] \ast g[n] = (f \ast g)[n] = \sum_{k=-\infty}^{\infty} f[k] \cdot g[n-k]
\end{equation}
```

`f[n]` ja `g[n]` on diskreetsed signaalid, mida hakatakse konvoleerima, `(f**g)[n]` on konvolutsiooni väljundsignaal. Sümbol `**` tähistab konvolutsiooni operatsiooni. Muutuja `k` on iteraator, mis käib läbi kõik teise signaali positsioonid esimese signaali suhtes.

Konvolutsioon on oluline algoritm digitaalses signaalitöötluses, kuna ühendab kolme väga tähtsat signaali: süsteemi sisend- ja väljundsignaali (`x[n]` ja `y[n]`) ning impulsskostet (*impulse response*) `h[n]`. Kui teame süsteemi impulsskostet, saame konvolutsiooni kasutades luua süsteemi väljundi mistahes sisendsignaalile: `y[n] = x[n] ** h[n]`.

Lihtsustatult tähendab see, et kui teame, kuidas süsteem mõjutab sisendsignaali iga sämplit, saame leida väljundi, mis tekib, kui süsteemi sisendisse anda järjestikku mitmeid sämpleid (signaal). Selle lähenemise kinnistamiseks implementeerime selles ülesandes konvolutsiooni, kasutades niinimetatud sisendsignaali algoritmi.

Sisendsignaali algoritmi vaatepunkt konvolutsioonist näitab, kuidas iga punkt sisendsignaalist mõjutab mitut punkti väljundsignaalist. Teisisõnu, leiame eraldi igale sisendsignaali punktile avaldatud mõju ja neid summeerides leiame korraga kogu väljundi. Rohkem informatsiooni leiab selle algoritmi kohta https://www.dspguide.com/ch6/3.htm.

Ülesande kirjeldus

Ülesandes on lubatud kasutada NumPy funktsioone massiivide loomiseks, omavaheliseks liitmiseks jne. Kuna tegeleme konvolutsiooni implementeerimisega, siis ei ole lubatud selle alamosana kasutada olemasolevat konvolutsiooni operatsiooni.

Etapp 1 - Ühe impulsi mõju

Nagu eelnevalt mainitud, võib konvolutsiooni toimimist rakenduslikult vaadata kui sämplite ükshaaval süsteemi sisestamist, kus süsteem avaldab igale üksikule sisendile mingit mõju. See mõju

tähendabki aga seda, et ühe sisendsignaali sämpli jaoks esineb mõjutav signaal (impulsskoste) süsteemi väljundis vastaval kohal nii tugevalt, kui on selle sisendsämpli väärtus.

Antud etapi eesmärgiks ongi luua funktsioon, mis rakendab mõju ühele ainsale sisendsignaali sämplile. See vastab signaalile, millel on üksainus nullist erinev väärtus - eelmisest praktikumist tuttav impulss. Sarnaselt eelmisele korrale on meil vaja seega defineerida impulsi amplituud ning ajahetk (nihe).

Funktsiooni sisenditeks on:

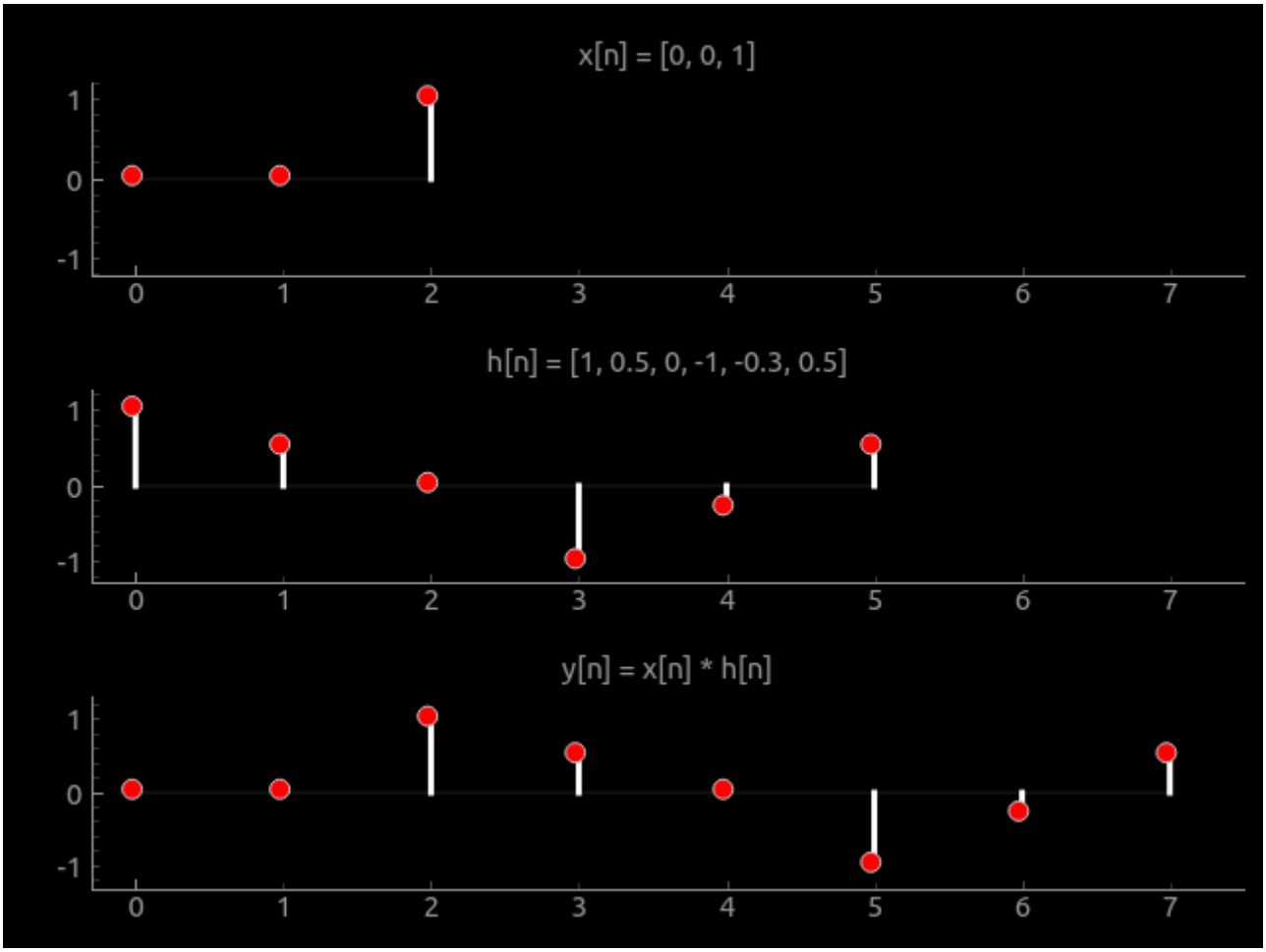
1. impulsi amplituud;
2. ajahetk, millal see impulss sisendsignaalis esines;
3. teine signaal (impulsskoste);
4. soovitava väljundsignaali pikkus.

Kuna ka sisendsignaali (pikkusega `n`) viimane sämpel (antud juhul ainus) saab mõjutatud kogu impulsskoste poolt (pikkus `m`), siis väljundsignaali pikkus peab olema piisav, et mahutada ära kogu tulemus.

Selle etapi lõpuks peab funktsioon töötama järgnevalt:

```
# Kontrollsisend nr 1.
>>> single_impulse_convolve(amplitude=0.5, time_index=3, imp_response=[1, 0.5, 0, -1,
-0.3, 0.5], length=9)
[0, 0, 0, 0.5, 0.25, 0, -0.5, -0.15, 0.25]
```

```
# Kontrollsisend nr 2 (kujutatud ka järgneval joonisel).
>>> single_impulse_convolve(amplitude=1, time_index=2, imp_response=[1, 0.5, 0, -1, -
0.3, 0.5], length=8)
[0, 0, 1, 0.5, 0, -1, -0.3, 0.5]
```



Joonis 1. Ühe sisendsignaali komponendi ja impulsskoste konvolutsioon.

Etapp 2 - Mõju summeerimine

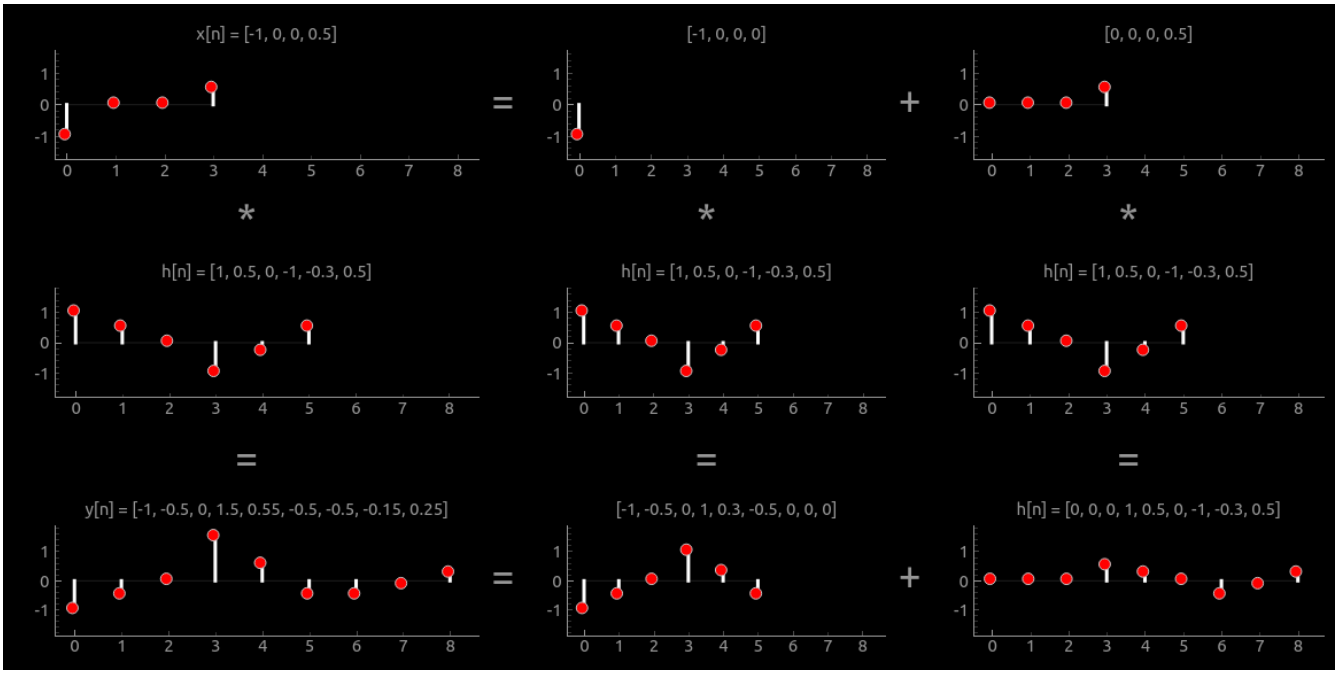
Kui meil on keerulisem sisendsignaal, millel on palju nullist erinevaid väärtusi, siis tuleb eelmises etapis implementeeritud loogikat rakendada kõigi sisendi elementide peal ja tulemused kokku summeerida.

Seega tuleb luua funktsioon, mis võtab sisendiks juba 2 suvalise pikkusega signaali ja tagastab konvolutsiooni tulemuse. See funktsioon peaks kasutama eelmises punktis loodud abifunktsiooni iga sisendsignaali punkti peal. Kõik üksikud tulemused peaksid olema lõpliku pikkusega (`m + n -1`), need tuleks omavahel kokku liita ja tulemuseks ongi kahe signaali konvolutsioon.

Ülesande käiku kirjeldavad järgnevad näited ja joonis.

```
# Kontrollsisend 1:
>>> input_side_convolution([2, 1, -3], [2, -2, 0, 3])
[ 4, -2, -8, 12,  3, -9]
# Arvutuskäik kasutades alamfunktsiooni näeb välja järgnev:
# single_impulse_convolve(amplitude=2, time_index=0, imp_response=[2, -2, 0, 3],
length=6)
# +
# single_impulse_convolve(amplitude=1, time_index=1, imp_response=[2, -2, 0, 3],
length=6)
# +
# single_impulse_convolve(amplitude=-3, time_index=2, imp_response=[2, -2, 0, 3],
length=6)
```

```
# Kontrollsisend 2 (kujutatud ka järgneval joonisel):
>>> input_side_convolution([-1, 0, 0, 0.5], [1, 0.5, 0, -1, -0.3, 0.5])
[-1, -0.5, 0, 1.5, 0.55, -0.5, -0.5, -0.15, 0.25]
# Arvutuskäik kasutades alamfunktsiooni näeb välja järgnev:
# single_impulse_convolve(amplitude=-1, time_index=0, imp_response=[1, 0.5, 0, -1,
-0.3, 0.5], length=9)
# +
# single_impulse_convolve(amplitude=0, time_index=1, imp_response=[1, 0.5, 0, -1,
-0.3, 0.5], length=9)
# +
# single_impulse_convolve(amplitude=0, time_index=2, imp_response=[1, 0.5, 0, -1,
-0.3, 0.5], length=9)
# +
# single_impulse_convolve(amplitude=0.5, time_index=3, imp_response=[1, 0.5, 0, -1,
-0.3, 0.5], length=9)
```



Joonis 2. Kahe signaali konvolutsioon jaotatuna osadeks vastavalt sisendsignaali algoritmile. Joonisel on graafikute vahele märgitud kehtivad seosed. Ülevalt alla on teostatud üksikud konvolutsioonid ühe ja sama impulsskostega (kuvatud teises reas) ja vasakult paremale on reas 1 näha sisendsignaali jaotust osadeks ja reas 3 konvoleeritud alamosade kombineerimist. Lõplik tulemus alumisel vasakpoolsel graafikul.

Näidake oma lahendust praktikumi juhendajale ja tehke commit!

2. Ülesanne: Konvolutsiooni implementatsiooni kiiruse hindamine

Sissejuhatus

Konvolutsiooni leidmiseks vajaminev tehete arv kasvab väga kiiresti signaali pikkuste suurenedes ja sellest tulenevalt ka aeg, mis kulub selle arvutamiseks. Kuna paljud signaalid, millega igapäevaselt tegeleme, sisaldavad miljoneid või enamaid sämpleid, on päris rakenduste puhul väga oluline, et konvolutsioon oleks implementeeritud võimalikult efektiivselt.

Paljud abioperatsioonid (tsüklite itereerimine, funktsiooni väljakutsed jms), mis eelmises ülesandes loodud lahenduses toimuvad saab aga madalamal implementatsioonitasemel ära jätta ja samuti saab protsessori võimekust paremini arvutusteks ära kasutada. Seda teeb näiteks NumPy teegis implementeeritud konvolutsiooni funktsioon.

Ülesande kirjeldus

Kasutades aluskoodi failis `yl2_1d_efektiivsus.py` tuleb luua programm, mis visualiseerimise teel võrdleb ajakulu eelmises ülesandes loodud konvolutsiooni implementatsiooni ja NumPy teegi optimiseeritud implementatsiooni vahel.

Antud koodis tuleb kõigepealt lisada abifunktsiooni aja mõõtmise funktsionaalsus ja seejärel määrata sobivad sisendite suurused. Maksimaalne sisend võiks aeglasema algoritmi puhul võtta ligikaudu 1 sekundi. Sisendite arv peaks olema piisav, et näha funktsiooni kuju, kuidas aeg sõltub sisendist.

Olge valmis tulemust juhendajale tõlgendama.

Näidake oma lahendust praktikumi juhendajale ja tehke commit!

3. Ülesanne: Heli konvolutsioon

Digitaalsest helist saab mõelda kui kvanditud analoogsignaalist, mis üldjuhul sisaldab sagedusi 20Hz kuni 20 000Hz. Need on sagedused, mis jäävad keskmise inimese kuulmisulatusse. Peale signaali digitaalseks muutmist on meil võimalik digitaalset signaali ilma kadudeta kopeerida, manipuleerida ning esitada. Järgnevas ülesandes kasutame konvolutsiooni, et siduda omavahel erinevaid helifaile ning impulsskosteid erinevatest asukohtadest üle maailma.

Süsteemi sisendiks saab `.wav` fail. Konvoleerides seda signaali mingisuguse ruumi impulsskostega, saame väljundsignaaliks heli selles asukohas esitatuna, ilma füüsiliselt kunagi seda heliklippi selles ruumis esitamata. Rohkem informatsiooni selle konvolutsiooni rakenduse kohta leiab otsides internetist *Convolution Reverb* või vaadates seda videot.

Selle praktikumi aluskoodi kaustas on mõned `.wav` failid. Faile, mille nimi algab eesliitega `sample_`, kasutame sisendsignaalidena ning faile, mille nimi liitega `ir_`, süsteemi impulsskostena. Failid, mille nimi lõpeb `_short` on sämplimissagedus on 8000 Hz ning ülejäänud failidel traditsiooniline 44100 Hz. Ülesandes tuleb valida failide kooslus, mis annab korrektse tulemuse.

Kasutades lingitud SciPy funktsioone looge programm, mis teeb järgnevat:

1. Loeb sisse kaks `.wav` faili - vabalt valitud sisendfaili ja impulsskoste
    a. Helifaili lugemiseks saab kasutada SciPy teegi vastavat funktsiooni
2. Normaliseerib need failid, kasutades iga faili maksimaalset väärtust, viies kõik väärtused vahemikku `[-1.0,1.0]`
    a. Normaliseerimiseks tuleks leida maksimaalne sämpli absoluutväärtus ja kogu sisend selle väärtusega läbi jagada (nt kui suurim sämpli väärtus on 0.8 ja vähim -0.9, tuleb kogu sisend jagada 0.9-ga)
3. Konvoleerib failid kokku kasutades NumPy teegis implementeeritud konvolutsiooni algoritmi ja normaliseerib tulemuse
4. Kirjutab normaliseeritud konvolutsiooni tulemuse kolmandasse `.wav` faili
    a. sarnaselt lugemisele on ka helifaili salvestamiseks SciPy teegis vastav funktsioon

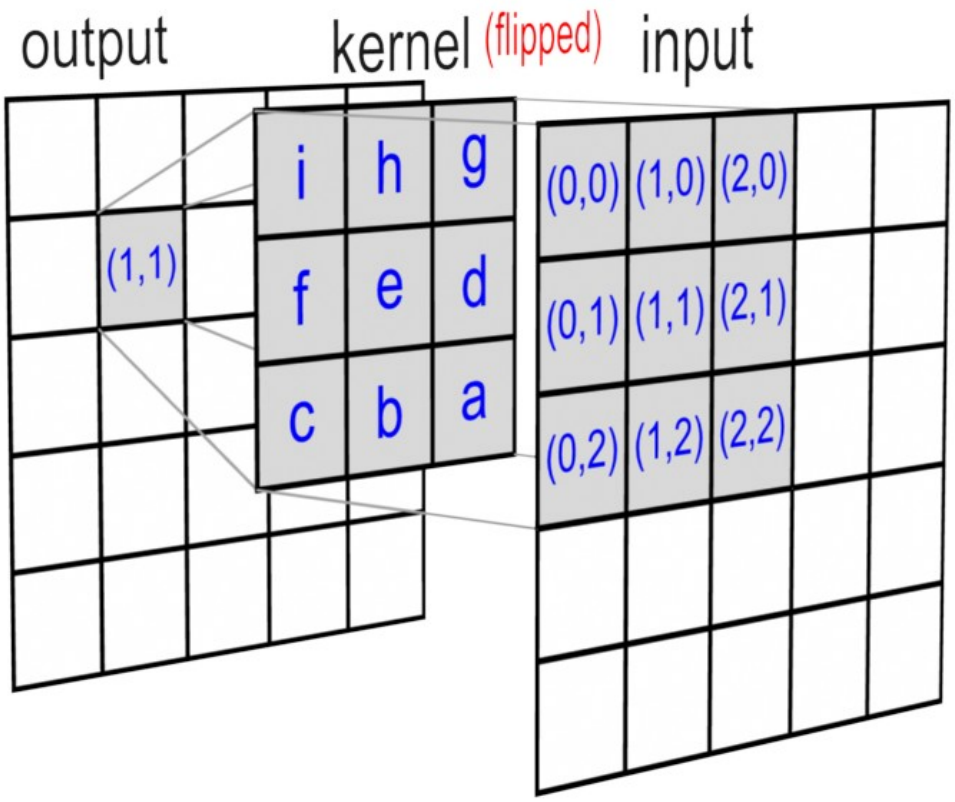
Ülesande arvestatuks saamiseks näidake oma konvolutsiooni tulemust ja kahte sisendfaili Audacitys. Seletage, kuidas selline väljund tekkis ning miks on väljundi kuju selline (keskenduda tuleks näiteks signaali lõpule, algusele ja kohtadele, kus sisendiks olev helifail on vaiksem).

Näidake oma lahendust ning selgitage tulemust praktikumi juhendajale ja tehke commit!

4. Ülesanne: 2D konvolutsioon pildil

Pilditöötluses on konvolutsioon kahe erineva suurusega maatriksi omavahel kombineerimine. Esimene neist on üldjuhul suur mustvalge pilt ning teine väiksem `M \times N` maatriks, kus `M` ja `N` on tihti võrdsed paaritud arvud. Seda maatriksit nimetatakse kerneliks.

Konvolutsioon viiakse läbi, libistades kernelit pildi kohal, alustades tavaliselt vasakult ülanurgast. Kernel saab pildil vabalt liikuda kohtades, kus see mahub täielikult pildi piiridesse (loogika, kuidas tegeleda äärtega, erineb vastavalt kasutusjuhule). Iga positsioon väljundis arvutatakse korrutades 180° pööratud kerneli iga lahtrit aluspildi vastava piksli väärtusega ning liites tulemuse kokku üheks arvuks. Selline lähenemine on väljundsignaali algoritm ja vastab otseselt valemi matemaatilisele kujule. Erinevalt esimesest ülesandest, kus kasutasime sisendsignaali algoritmi, leiame siin ühe väljundi elemendi korraga.



Joonis 3. Kahemõõtmeline konvolutsioon pildil. Tasub tähele panna, et sisendi vaadeldava piksliga läheb kohakuti kerneli keskpunkt, kuna enamustes pilditöötluse rakendustes tahame, et kernel mõjutaks pikslit sümmeetriliselt tema ümbruskonnas.

```
\begin{equation}
    y[1,1] = i \cdot x[0,0] + h \cdot x[1,0] + g \cdot x[2,0] + f \cdot x[0,1] + e
\cdot x[1,1] + d \cdot x[2,1] + c \cdot x[0,2] + b \cdot x[1,2] + a \cdot x[2,2]
\end{equation}
```

Matemaatilisel kujul on 2D konvolutsioon defineeritud järgnevalt:

```
\begin{equation}
    y[m,n] = x[m,n] \ast h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty}
x[i,j] \cdot h[m-i,n-j]
\end{equation}
```

`y[m,n]` on konvolutsiooni väljund, `x[m,n]` on algne pilt, `h[m,n]` on konvolutsiooni kernel. Valemi praeguse kuju puhul tähistab `m` veergusid ning `n` ridu.

Konvolutsiooni sisulisemaks mõistmiseks tasub lähemalt uurida selle taga olevat matemaatikat. Ühte head näidet internetis saab näha siit.

2D konvolutsiooni implementeerimine

Järgnevaks praktikumiülesandeks on kirjutada funktsioon, mis võtab sisendiks kaks kahemõõtmelist massiivi. Üks neist on sisendsignaal ja teist kasutame kernelina. Kasutades neid argumente, teostab funktsioon 2D konvolutsiooni ning tagastab väljundi. Võib eeldada, et pilt ei ole kernelist väiksem ning kerneli mõõtmed on paarituarvulised. Loogika, kuidas tegeleda äärtega, kus konvolutsiooni kernel ei mahu täielikult algsesse pilti, on teie otsustada.

Ülesande lahendamiseks võib kasutada alamkäskudena NumPy töövahendeid, kuid otse konvolutsiooni operatsiooni kasutamine on keelatud.

Peale algoritmi implementeerimist peab see töötama järgnevalt:

```
# 2D konvolutsiooni näide
img = np.array((
    [ 1, 3, 3, 2],
    [ 7, 9, 8, 9],
    [ 7, 8, 6, 7],
    [ 8, 9, 9, 7]), dtype="int")

kernel = np.array((
    [-1, -2, -1],
    [ 0,  0,  0],
    [ 1,  2,  1]), dtype="int")

output = convolve(img, kernel)

# print(output), kui pildi ümber luuakse regioon väärtustega 0
[[-23. -33. -34. -26.]
 [-17. -19. -16. -13.]
 [ -2.  -2.   0.   3.]
 [ 22.  29.  27.  20.]]

# print(output), kui konvolutsiooni tehakse ainult siis, kui kernel mahub
täielikult pildi piiridesse
[[  0.   0.   0.   0.]
 [  0. -19. -16.   0.]
 [  0.  -2.   0.   0.]
 [  0.   0.   0.   0.]]
```

Näidake oma lahendust praktikumi juhendajale ja tehke commit!

5. Ülesanne: Pildi udustamine, servade tuvastamine

Selleks, et siduda kahemõõtmelist maatriksit pildi formaadiga, saab kasutada OpenCV teeki. Selle abil on võimalik pildi sisselugemine failist NumPy massiiviks ja vastupidi. Samuti sisaldab teek funktsionaalsust sobival kujul olevate maatriksite kuvamiseks visuaalselt pildina. Antud ülesande jaoks olulised käsud on välja toodud ülesande aluskoodis ja täpsemalt saab lugeda OpenCV ametlikust dokumentatsioonist.

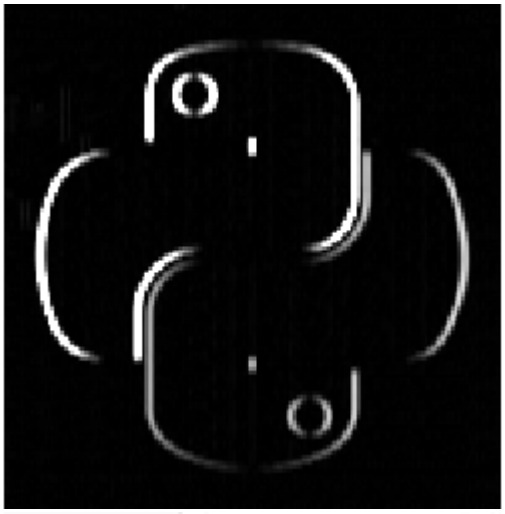
Kui cv2.imshow() näitab teile imelikku tulemust, soovitame uurida teie pildi andmetüüpi.

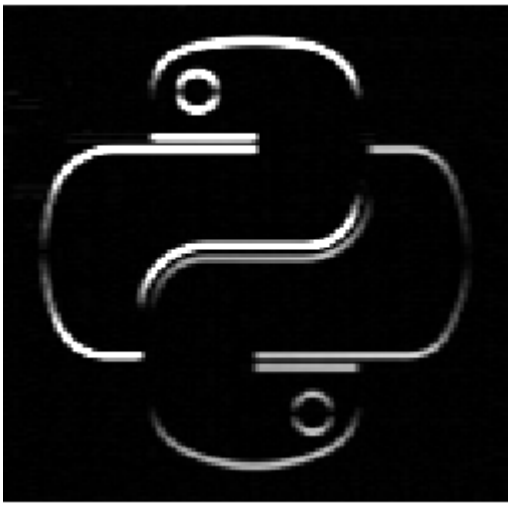
Järgnevaks ülesandeks on, kasutades eelnevalt loodud konvolutsiooni funktsiooni, implementeerida järgnevad funktsioonid `box_blur()`, `gaussian_blur()`, `sobel_x_edge()`, `sobel_y_edge()` ja `laplacian_edge()`. Funktsioonid võivad võtta nii mitu sisendit kui oluliseks peate. Kõik vajalikud kernelid on saadaval aine repositooriumis käesoleva praktikumi aluskoodi kaustas.

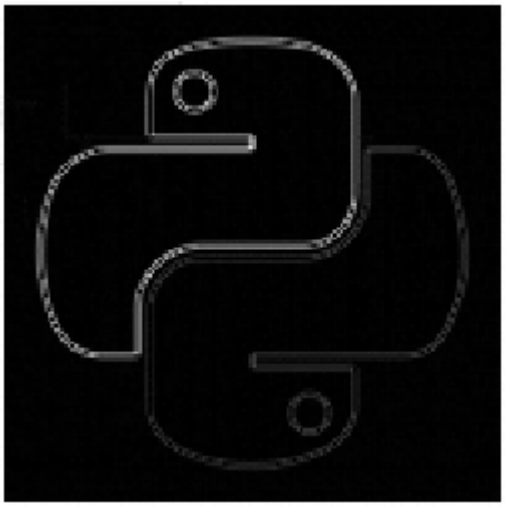
Ülesande lahendamisel tasub tähele panna, et pildi puhul on oluline, et väljund püsiks õige andmetüübi raamides. Seega tuleb veenduda, et lõpptulemuses jääksid väärtused 0 ja 255 vahele. Selle saavutamiseks kasutame sageli normaliseeritud kerneleid või jagame tulemuse kerneli elementide summaga. Ebaühtlaste kaaludega kernelite puhul tähendab see ka seda, et tulemust võib olla vaja piirata.

Piltidel toodud funktsioonide väljundid vasakult paremale:

1. tavaline udustamine 11×11 kerneliga;
2. Gaussiani udustamine 5×5 kerneliga;
3. Sobel X;
4. Sobel Y;
5. Laplacian servade tuvastamine.







Näidake oma lahendust praktikumi juhendajale ja tehke commit!

4. Kodutöö

1. ülesanne

Kasutades praktikumis loodud funktsioone ning ette antud koodi, tuleb luua, konvoleerida ja kujutada graafiliselt kahe kastsignaali konvolutsioon:

1. Looge sobivad sisendsignaalid.
    a. Kastsignaali loomiseks saab kasutada vastavat SciPy funktsiooni.
    b. Kastsignaali minimaalsed väärtused on nullid ning maksimaalsed võite ise valida.
    c. Samuti võib ise valida signaali pikkuse, aga see peab sisaldama vähemalt ühte täisperioodi.
2. Konvoleerige signaalid ja kuvage graafiliselt nii lähtesignaalid kui tulemus.
3. Selgitage, miks tekib just selline väljund.

Eeldatav vastus:

- sisaldab linki repositooriumis asuvale kommenteeritud koodifailile, mis teostab ülesandes nõutud konvolutsiooni;
- selgitab lühikese lõiguga väljundi kuju;

Loodud kood tuleb laadida oma repositooriumi ning vastavad failid aruandes linkida!

2. ülesanne

Selgitage ja demonstreerige konvolutsiooni kommutatiivsust ja assotsiatiivsust. Demonstreerimisel tuleb ise valida sobivad signaalid. Kasutada võib nii enda loodud kui ka NumPy teegist leitavat konvolutsiooni.

Eeldatav vastus:

- sisaldab lauset, milles seisneb konvolutsiooni kommutatiivsus ja esitab vastava matemaatilise võrdsuse;
- sisaldab lauset, milles seisneb konvolutsiooni assotsiatiivsus ja esitab vastava matemaatilise võrdsuse;
- sisaldab linki repositooriumis asuvale kommenteeritud koodifailile, mis ise valitud sobivate näidete toel kinnitab kommutatiivsuse ja assotsiatiivsuse kehtivust;

Loodud kood tuleb laadida oma repositooriumi ning vastavad failid aruandes linkida!

3. ülesanne

1. Selgitage sobivate näidete abil, kuidas Sobeli kernelid tuvastavad servi vertikaal- ja horisontaalsuunas.
    a. Mõelge selle peale, kuidas selline kerneli paigutus reageerib ühtlase regiooni ja serva peale.
2. Leidke internetist veel mõni huvitav kernel, demonstreerige selle tööd enda valitud pildil ning seletage lühidalt kerneli taga peituvat loogikat.
    a. Lisage pilt kerneli väljundist oma kodutöö faili ning selle põhjal seletage kerneli tööd.

Eeldatav vastus:

- kirjeldab Sobeli kerneli tööd toetudes näidetele, mis juhtub erinevate sisendite korral;
- toob välja ühe praktikumis mitte kasutatud kerneli kuju;
- viitab repositooriumis asuvale koodile (link õigele failile gitlab.ut.ee keskkonnas), mis demonstreerib leitud kerneli tööd.

Ülesandes loodud kood tuleb laadida oma repositooriumi ning vastavad failid aruandes linkida!

4. ülesanne

Küllap olete nüüdseks tähele pannud, et praktikumis loodud 1D ja 2D konvolutsiooni algoritmid pole just kõige kiiremad. Programmi täitmisaeg pole väikese andmemahu juures tänapäeval eriti tajutav, kuid üldjuhul tehakse signaalitöötlust andmetega, millel on tuhandeid (kui mitte kümneid tuhandeid) andmepunkte. Enamikel aluskoodi kaustas olevatel `.wav` failidel on sämplimise kiiruseks 44.1 kHz, mis tähendab, et tavalist 4-minutilist laulu töödeldes on meil `4 * 60 * 44100 (10 580 000)` andmepunkti. Arvutage, mitu liitmis- ja korrutamistehet tuleb teha sellise loo konvoleerimiseks samal sämplimissagedusel oleva 20-sekundilise impulsskostega.

Eeldatav vastus:

- sisaldab arvutuskäiku;
- toob iga tehte kohta välja, millisest sisendsignaali algoritmi osast see tuleneb;
- annab 2 lausega hinnangu lõplikule väärtusele.