Sissejuhatus, signaali kujutamine

Sisukord

Aastal 2021 on vastutav juhendaja: Kristo Allaje <kristo.allaje@ut.ee> 2021-02-09|16:39:31 +0200

1. Ülevaade

Juhendist on olemas ka PDF versioon.

See praktikum tutvustab ja kordab programmeerimise võtteid ja tööriistu, mis on olulised signaalide töötlemisel. Samuti annab see esmase ülevaate signaalidest ja võimalustest signaale sisse lugeda ja salvestada.

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

1. Python3
2. Arduino IDE koos Arduino Nanoga
3. PyQtGraph
4. PySerial
5. NumPy

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

2. Kohustuslikud ülesanded

0. ülesanne: Praktikumiks valmistumine

Repositooriumi ülesseadmine ja esimene commit

Soovitame GitLabi kasutaja luua enne esimesse praktikumi tulemist!

Praktikumide koodi haldamine hakkab toimuma kasutades Git'i. Selleks kasutame Tartu Ülikooli GitLab'i, kuhu on võimalik siseneda kasutades Tartu Ülikooli autentimist. Kursuse repositooriumid on kõigi jaoks genereeritud ja sellele ligipääsu saamiseks on oluline, et oleks vähemalt korra oma kasutajaga keskkonda sisse logitud, kasutades varianti `sign in with` **[shibboleth]**.

HTTPS protokolli kasutades on vaja luua oma kasutajale teine autentimisvahend - GitLab'i kohalik parool (vajalik klassiarvutiga töötamisel). Peale GitLab'i sisenemist liikuda paremal üleval nurgas asuvast menüüst **Settings › Password** ja luua omale uus parool. Klassiarvutiga töötamiseks on vaja teada ka oma GitLab'i kasutajanime. Selle leiab GitLab'i seadetest **Settings › Account › Change username** juurest.

Nüüd on võimalik repositoorium kloonida kasutades käsku `git clone [aadress]`. Kloonimiseks vajaliku aadressi leiab, kui liikuda aine repositooriumisse ja sinise *clone* nupu alt kopeerida aadress väljalt `Clone with HTTPS`.

GitLab'i kasutamise õpetuse leiab siit.

Ülesande arvestatuks märkimiseks on vaja kloonida repositoorium kohalikku arvutisse ja luua seal omanimeline haru, mille nimi on kujul: `tudeng/Eesnimi-Perenimi`. Peale haru loomist tuleb see lükata serverisse kasutades käsku `git push -u origin tudeng/Eesnimi-Perenimi`.

Pythoni virtuaalkeskkond

Selle aines kasutatakse Pythoni virtuaalkeskonda, milles on olemas kõik vajalikud teegid aine praktikumide jaoks. See on leitav kasutaja kodukataloogist, kaustas nimega `dsp-env`.

Virtuaalkeskkonna aktiveerimiseks tuleb avada terminal, navigeerida kasutaja kodukataloogi ning jooksutada järgnevat käsku `source dsp-env/bin/activate`. Nähes terminalis järgnevat eesliidest `(dsp-env) kasutaja@arvuti` võib olla kindel, et virtuaalkeskonna aktiveerimine õnnestus. Kõik edasine toimub tavapäraselt ning kasutades käsku `python3 minu_kood.py` jooksutatakse kood kasutades virtuaalkeskkonnas olevaid teeke. Virtuaalkeskkonna välja lülitamiseks tuleb terminali anda käsk `deactivate`.

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

# Virtuaalkeskkonnas koodi jooksutamine
(dsp-env) kasutaja@arvuti:~$ python3 minu_kood.py

#Virtuaalkeskkonna välja lülitamine
(dsp-env) kasutaja@arvuti:~$ deactivate
```

Samuti on õpilaste repositooriumis olemas fail nimega `requirements.txt`, mida uuendatakse jooksvalt kursuse edenedes. Selles on kirjas kõik vajalikud teegid, mida kursuse jooksul kasutatakse. See annab võimaluse luua Pythoni virtuaalne keskkond enda isiklikus arvutis, kus on olemas kõik vajalik praktikumide ülesannete lahendamiseks. https://docs.python.org/3/tutorial/venv.html#creating-virtual-environments

```
# Oma enda loodud virtuaalkeskkonnas vajalike teekide paigaldamine
(dsp-env) kasutaja@arvuti:~$ python3 -m pip install -r requirements.txt
```

1. ülesanne: Pythonis programmeerimise kordamine

Sissejuhatus

Ülesande eesmärgiks on meelde tuletada programmeerimise süntaks keeles Python. Ülesande käigus saab harjutada programmi sellise kujuga valemite ümber kirjutamist, mille sarnaseid esineb sageli signaalitöötluses. Esimesed näited on lihtsamad ning järgnevad järjest keerulisemad. Implementeerimise käigus proovige ka aru saada, mida need valemid sisuliselt teevad.

Programmeerimisel on tihtilugu ühes failis implementeeritud funktsioone vaja hiljem kasutada ka teistes failides. Terve funktsiooni kopeerimine uude faili võib aga olla tülikas, samuti muudab see programmifaili mahukamaks ja seega on failis olevat koodi raskem hallata. Selle vältimiseks saab varasemalt teises failis defineeritud funktsiooni ka lihtsalt programmi importida, kasutades sarnast käsku nagu sisseehitatud teekide importimisel.

Kuna eelnevas failis võib olla ka muud sisu, näiteks nende funktsioonide testväljakutsed, siis on oluline kindlaks teha, et neid väljakutseid uude faili üle ei tooda. Selleks on võimalik funktsioonid kirjutada täiesti eraldi faili või kasutada Pythoni sisseehitatud muutujat `__name__`. See muutuja saab programmis A väärtuse `__main__` siis, kui programmi A jooksutatakse põhifailina, aga mitte siis, kui programmi A fail on imporditud programmi B faili. Lisades nüüd kogu testkoodi tingimuslausesse, mida jooksutatakse ainult siis, kui tegu on põhiprogrammiga, saabki soovitud tulemuse. Lisainfot koos näidetega on võimalik lugeda sellelt lingilt.

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-

# See osa loetakse sisse nii siis, kui käivitatakse otse see programmifail, kui ka siis, kui see fail on imporditud mõnda teise faili.
# Siin defineerida funktsioonid, mida saab importida ka teistesse programmidesse
def foo(x):
  return x

def bar(x):
  return x

# Väga levinud tava on faili käivitamisel panna jooksutatav kood main nimelisse funktsiooni.
# Paljudes teistes keeltes on samuti main funktsioon, kus kohast algab koodi täitmine.
# https://realpython.com/python-main-function/
def main():
  x = 5
  y = foo(x)
  z = bar(y)

if __name__ == "__main__":
  # See osa loetakse sisse ainult siis, kui .py fail käivitatakse otse.
  # Kui see fail on imporditud mõnda teise programmifaili, siis teise faili käivitamisel siin olevat osa ei loeta.
  main()
```

Ülesande kirjeldus

Ülesande läbimiseks on vaja implementeerida järgnevad valemid 1-9 Pythoni funktsioonidena. Funktsioonide sisenditeks on kõik muutujad, millest sõltub tulemus, ja valemites 7-9 ka sisendjärjend. Sisendparameetrite nimed funktsiooni kirjelduses peavad vastama muutujanimedele avaldises ja funktsiooni nimes peab kajastuma implementeeritava valemi number. Funktsioon peab tagastama valemi rakendamise tulemusel saadud väärtuse vastava(te) sisendi(te) korral.

Mistahes suure sigma notatsiooni kasutava summa implementeerimisel on kasulik mõelda tsüklitele. Suure sigma notatsiooni tähendusest koos näidetega saab lähemalt lugeda sellest ingliskeelsest Vikipeedia artiklist ja kõrgema matemaatika konspektist leheküljel 7.

Oma implementatsiooni kontrollimiseks on võimalik valemite 1-6 puhul kasutada internetikalkulaatoreid nagu näiteks WolframAlpha. Lihtsaim viis selleks on kopeerida juhendist valemi LaTeX kuju, mille leiab, kui teha paremklõps valemi peal ja valida avanenud menüüst "Show Math As" → "TeX Commands". Selle valemi saab kopeerida otse WolframAlpha avalehel olevasse vormi. Enne arvutamist tuleb valemisse sisse panna sisendväärtused, millega funktsiooni testida ja WolframAlpha kuvab kas täpse vastuse või ligikaudse lähendi (`Decimal approximation`).

Implementeeritavad valemid

Suure sigma notatsiooni puhul on ülemine väärtus kaasa arvatud.

$$\sum_{n=0}^{N} \frac{1}{n!} \quad (1)$$

$$\sum_{k=0}^{K-1} \frac{r^k}{a} \quad (2)$$

Python võimaldab funktsiooni parameetreid väärtustada ka nimeliselt. Sellisel juhul ei pea hoolega jälgima parameetrite ette andmise järjekorda.

Järgnevas valemis tuleb mõelda ka sellele, mis juhtub, kui kasutaja kutsub funktsiooni välja sisendiga $R < 5$ ja $V < 1$. Matemaatiliste konstantide jaoks soovitame kasutada NumPy nimelist teeki, mille dokumentatsioon konstantide osas on leitav sellelt lingilt.

$$\sum_{r=5}^{R} \sum_{v=1}^{V} (\frac{1}{\sin 2r} + \frac{2v}{3\pi}) \quad (3)$$

Järgnevas valemis tähistab e Euleri arvu. Sarnaselt eelmisele valemile tuleb ka siin mõelda sellele, milliste sisenditega saab antud valemi tulemit arvutada.

$$\sum_{x=1}^{25} \sum_{b=a}^{B} \sum_{m=3}^{M} (e^b + m \cdot x) \quad (4)$$

Sarnaselt suurele sigmale, mis tähistab samakujuliste avaldiste liitmist, tähistab suur pi avaldiste korrutamist.

$$\prod_{i=1}^{A} (ki + \frac{1}{2k}) \quad (5)$$

$$\prod_{c=k}^{C} \sum_{d=l}^{D} (\frac{c}{d}) \quad (6)$$

Järgnevate valemite üheks sisendiks on funktsioon, mis on tähistatud kui f või g. Kui olulised on ainult osad funktsiooni väärtused, siis saab sellest mõelda kui järjendist. Seega on järgnevate funktsioonide (üheks) sisendiks järjend, mille pikkus on N või M. Seega ei ole vaja N (ega M) eraldi sisendina anda. Indekseerimine kujul $f[n]$ viitab, et järjendist f kasutatakse elementi kohal n. Pidage meeles, et indekseerimine algab 0-st.

$$\sum_{n=0}^{N-1} (3f[n] - 7) \quad (7)$$

$$a \sum_{n=1}^{N-1} (f[n-1] + f[n]) \quad (8)$$

Kui sisendiks on kaks funktsiooni, siis pikkus N viitab järjendi f pikkusele ja M järjendi g pikkusele.

$$\sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (f[n]\cos(\pi g[m])) \qquad (9)$$

Ülesande arvestatud saamiseks tuleb juhendajale näidata kõiki valemite implementatsioone ja jooksutada neid juhendaja küsitud sisenditega.

Näidake oma lahendust juhendajale ning tehke commit!

2. ülesanne: Numerical Python

Numerical Python ehk NumPy on Pythonis teaduslike arvutuste jaoks üks põhipakette. See on ülimalt kiire ning laialdaste võimalustega tööriist, mida kasutakse Pythonis igal pool, kus on vaja teha midagi kiiresti. See on üks selle kursuse põhitööriistu ning soovitame tungivalt end sellega kurssi viia, kui te pole seda teeki enne kasutanud. Hea ülevaate NumPy võimalustest saab lugedes seda NumPy õpetust, millest allpool toome välja mõned tähtsamad näited.

NumPy massiivi statistikud ja *broadcasting*

Allpool on välja toodud mõned kõige sagedamini kasutatavad operatsioonid, mida NumPy massiive kasutades vaja läheb.

```
>>> a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> a.shape
(10,)
>>> a.size
10
>>> a.sum()
55
>>> a.min()
1
>>> a.max()
10
```

NumPy võimaldab massiividel teostada lihtsat standardset matemaatikat, samuti keerukamat lineaarset algebrat, näiteks massiivide korrutamist. Rohkem informatsiooni selle võimaluse kohta leiab otsides infot NumPy dokumentatsioonist *broadcasting* kohta.

```
>>> data = np.array([1.0, 2.0])
>>> data * 1.6
array([1.6, 3.2])

>>> data1 = np.array([[1, 2], [3, 4]])
>>> data2 = np.array([[1, 2], [3, 4]])
>>> data1.transpose()
array([[1, 3],
       [2, 4]])
>>> data1 * data2.transpose()
array([[ 1,  6],
       [ 6, 16]])
```

Maskid

NumPy maskide abil on võimalik kiiresti filtreerida massiivis olevaid andmeid. Maskide abil on võimalik massiivist soovitud kriteeriumile vastavaid elemente leida ning siis vastavalt vajadusele muuta.

```
>>> array = np.random.randint(0,100,size=(1,10))
>>> array
array([[20, 20, 65, 57, 14, 36, 40,  0, 64, 16]])
>>> mask = array > 50
>>> array[mask]
array([65, 57, 64])
>>> array[mask] = -1
>>> array
array([[20, 20, -1, -1, 14, 36, 40,  0, -1, 16]])
```

Ülesanded

$$\sum_{r=1}^{R}\sum_{v=1}^{V}(2rv) \qquad (10)$$

Ülesande arvestatud saamiseks tuleb juhendajale näidata järgnevat:

1. Implementeerige ülal olev valem kasutades Pythoni `for` tsükleid.
2. Implementeerige ülal olev valem kasutades NumPy poolt pakutavat vektoriseeringut. Ülesande lahenduses ei tohi kasutada tsükleid ning kontrollige väljundit võrreldes seda mitte vektoriseeritud lahendusega.
   a. Numpy.newaxis't või numpy.reshape saab kasutada olemasoleva järjendi dimensioonaalsuse suurendamiseks.
   b. Topelt summast tuleks mõelda kui maatriksist.

```
\begin{equation}
  \begin{bmatrix}
  2r_{1}v_{1} & 2r_{1}v_{2} & ... & 2r_{1}v_{V-1} & 2r_{1}v_{V}\\
  2r_{2}v_{1} & 2r_{2}v_{2} & ... & 2r_{2}v_{V-1} & 2r_{2}v_{V}\\
  ... & ... & ... & ... & ... \\
  2r_{R-1}v_{1} & 2r_{R-1}v_{2} & ... & 2r_{R-1}v_{V-1} & 2r_{R-1}v_{V}\\
  2r_{R}v_{1} & 2r_{R}v_{2} & ... & 2r_{R}v_{V-1} & 2r_{R}v_{V}\\
  \end{bmatrix}
\end{equation}
```

3. Looge funktsioon, mis loob kasutades np.random.randint maatriksi suurusega (4, 10), mille elemendid ulatuvad 0st 100ni. Peale maatriksi loomist prindib funktsioon välja algse maatriksi ja maatriksi veerud, mille keskmine väärtus on suurem terve maatriksi elementide keskmisest.
   a. Leidke veergude keskmine ning kasutades seda looge mask, mis näitab, missuguse veeru keskmine on suurem maatriksi keskmisest väärtusest.
   b. Loodud maski kasutades väljastage väärtus igast maatriksi reast, kus loodud maski väärtus on tõene.

Näidake oma lahendust juhendajale ning tehke commit!

Kodus tuleb lahendada esimene koduülesanne

1. Lisage oma programmi võimalus mõõta aega, kasutades aluskoodis demonstreeritud timeit funktsionaalsust. Võrrelge implementeeritud valemi naiivse ja vektoriseeritud kuju töökiirust erinevate sisendite korral ning kujutage tulemusi tabelis.

   Eeldatav vastus:

   - sisaldab vähemalt viie erineva sisendiga sooritatud mõõtmiste tulemusi, kusjuures vähima ja suurima sisendi vahe peab olema vähemalt tuhat korda;
   - võrdleb ning kirjeldab saadud tulemusi lühidalt.

2. Uurige NumPy poolt pakutavaid võimalusi ning kirjutage vähemalt ühest funktsioonist, mis võiks teid tulevikus aidata signaalide uurimisel.

   Eeldatav vastus:

   - kirjeldab lühidalt, mida valitud funktsioon teeb ning miks te just selle valisite;
   - sisaldab linki vastava funktsiooni dokumentatsioonile NumPy koduleheküljel;
   - demonstreerib paari koodireaga funktsiooni tööd.

3. ülesanne: Graafikud

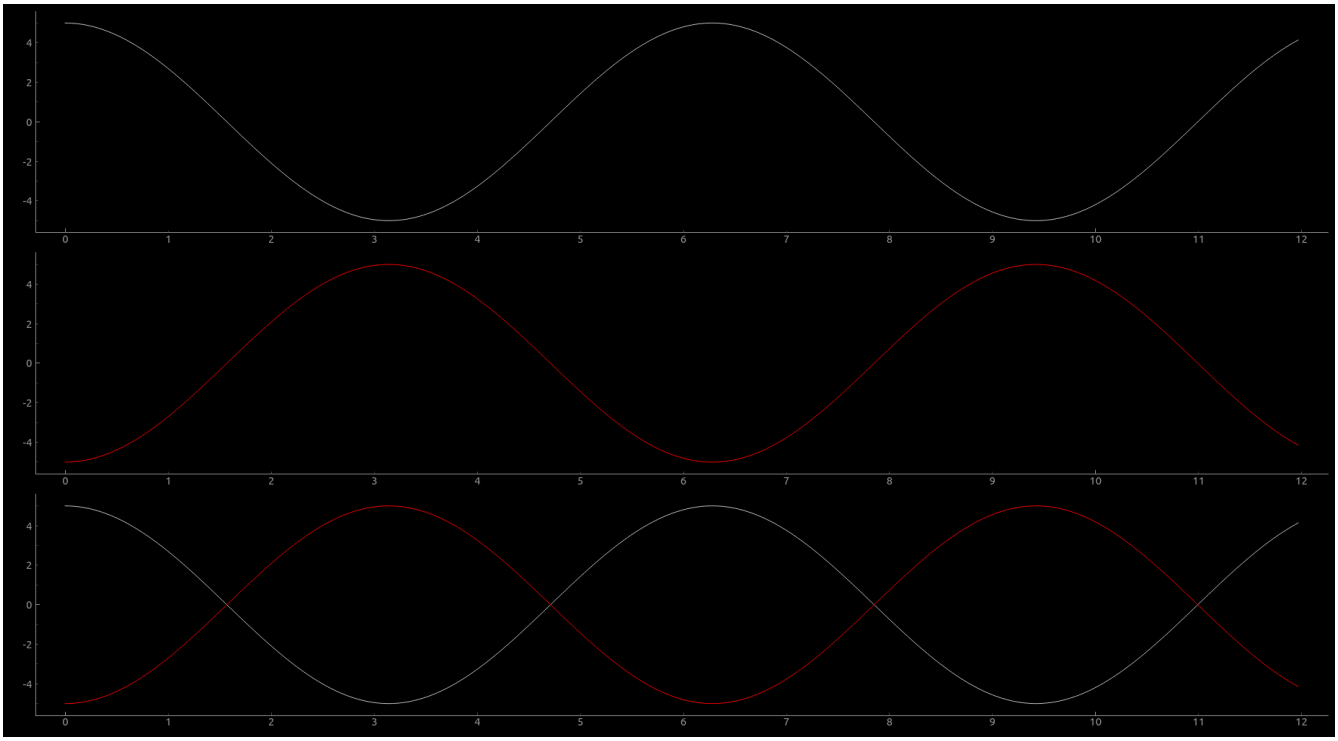
Sissejuhatus

Selles aines, nagu nimigi viitab, tuleb palju tegeleda erinevate signaalidega. Enamikku signaalidest on võimalik visualiseerida ning üks enim levinud viise selleks on kasutada graafikuid. Järgneva ülesande eesmärk on vaadata, kuidas joonistada Pythonis graafikuid, mis võimaldavad signaali kuvada selliselt, et see oleks vaataja jaoks võimalikult informatiivne.

Ülesande kirjeldus

Selle ülesande käigus saab tutvuda ühe populaarse ja palju võimalusi sisaldava teegiga PyQtGraph, mis on üks paljudest graafikute joonistamist võimaldavatest teekidest Pythonis. Ülesande eesmärk on vaadelda eri liiki graafikute joonistamise võimalusi, graafikul olevate joonte ja punktide eristamise võimalusi, mitme graafiku kõrvuti kuvamise võimalusi ja legendide koostamist.

Alustamise lihtsustamiseks on teie aine repositooriumis fail nimega `yl3.py`. See fail kujutab lihtsaid siinus- ja koosinusfunktsioone, et välja tuua PyQtGraph'i kasutamise põhitõed. Esimese asjana tuleb veenduda, et antud programmilõik töötab, ja seejärel aru saada, mida see teeb.



Joonis 1. Aluskoodi jooksutamisel saadav graafik.

Programmi väljundit vaadates võib vaatlejas tekkida segadus, kuna joonisel puudub info, mille abil graafikutel kujutatut mõtestada. Esimene asi, mis näidisel puudub, on telgede suuruste nimetused. Signaalidega töötamisel on sageli graafikute x-teljel aeg või sagedus; y-teljel temperatuur, pinge, asukoht või võimendus. Samuti on neil suurustel alati olemas ühikud, mis peavad olema telgede juurde märgitud, et vältida graafiku mitmetimõistmise võimalust.

Järgmiseks oletame, et antud signaalinäidete puhul t on aeg sekundites ja y-teljel kuvatakse pendli hälve tasakaaluasendist sentimeetrites sel ajavahemikul. Lisaks võiks olla igal graafikul ka mainitud, mis joonega on tegu. See on eriti oluline juhul, kui ühel graafikul on mitu joont (nagu

kolmas graafik näidisprogrammis). Selleks otstarbeks saab graafikutele lisada legendi käsuga `plot.addLegend()`. Peale graafiku sildistamist on võimalik anda `plot` funktsiooni väljakutsel nimeline parameeter `name="joone_nimi"`. Täpsemalt saab legendi kasutamisest lugeda PyQtGraph dokumentatsioonist või otsides internetist.

Lugedes dokumentatsiooni on näha, et igale graafikule saab lisada ka pealkirja kasutades `plot.setLabels()`. Sedasi saab kasutajale teada anda, mis graafikuga üleüldse tegemist on. Samasugune graafik võib väljendada nii radiaatori temperatuuri ajas kui ka pendli positsiooni võnkumisel, seega on oluline teada anda, mis on graafikul kuvatud.

Järgnevat ülesannet lahendades võib teid aidata järgnev koodijupp.

```
1 import pyqtgraph.examples
2 pyqtgraph.examples.run()
```

Iseseisvalt, dokumentatsiooni kasutades, lahendada järgnev koondnimekiri alamülesannetest. Ülesande arvestatud saamiseks tuleb esitada kõik alampunktid korraga:

1. Graafikutel peavad olema sisu kirjeldavad pealkirjad;
2. Graafikutel peavad olema korrektsed legendid, mis kajastaksid funktsiooni, mida graafikul olev joon esitab;
3. Kolmele graafikule tuleb lisada telgedele suurused koos ühikutega;
    a. X-telg tähistab aega 0st 12 sekundini.
    b. Y-telg tähistab pendli nihet tasakaaluasendi suhtes sentimeetrites.
4. Graafikute taustal tuleb kuvada ruudustik, et graafiku pealt oleks väärtusi lihtsam lugeda.

Näidake oma lahendust juhendajale ning tehke commit!

Kodus tuleb lahendada teine koduülesanne

1. Uurige oma loodud graafikuid. Mis on nendel graafikutel olevate siinus- ja koosinusfunktsioonide perioodid ja sagedused? Millised oleksid periood ja sagedus, kui telgedel kujutatud ühikud ei oleks mitte sekundid, vaid minutid?

   Eeldatav vastus:

   - kirjeldab lühidalt probleemile lähenemist ja arvutuskäiku;
   - toob eraldi välja arvutuskäigus kasutatud valemid.

2. Otsige PyQtGraph'i dokumentatsioonist veel vähemalt 2 erinevasisulist funktsiooni, mis tunduvad vajalikud ja millega saab muuta graafiku teatud parameetreid. Proovige neid funktsioone programmis kasutada ning lisage tulemustest pildid aruandesse. Kirjeldage aruandes oma sõnadega, mida need funktsioonid teevad ja tooge välja olukorrad, kus need kasulikuks osutuksid, koos põhjendusega.

   Eeldatav vastus:

   - kirjeldab lühidalt, mida valitud funktsioon teeb ning miks te just selle valisite;
   - sisaldab linki vastava funktsiooni dokumentatsioonile PyQtGraph koduleheküljel;
   - demonstreerib paari koodireaga funktsiooni tööd;
   - sisaldab pilti funktsiooni rakendamise tulemusest;
   - orignaalmõõdus pildid demonstreeritavatest graafikutest on lisatud teie repositooriumi.

4. ülesanne: Signaali lugemine Arduino abil

Sissejuhatus

Kui eelnevas ülesandes oli signaal juba eelnevalt defineeritud, siis nüüd tuleb vaatluse alla reaalajas sisendina saadav signaal. Lisaks sellele on siin ülesandes kasutatav signaal pärit välismaailmast ja seega tuleb see esmalt sisse lugeda ehk tuleb sooritada mõõtmine.

Ülesande kirjeldus

Signaal, mida selles ülesandes mõõta tuleb, on esitatud pingena. Seega on vaja seadet, mis suudaks sellise signaali võimalikult hästi tuua digitaalsesse maailma. Digitaalsel kujul oleva signaaliga oskavad töötada meie tavapärased arvutid. Selles praktikumiülesandes kasutame signaali sisse lugemiseks Arduino Nano arendusplaati, millel on olemas ka analoog-digitaalmuundur (ingliskeelne lühend ADC). ADC abil on võimalik muuta pinge väärtusi kokkuleppelisest nullist kuni maksimaalse pingeni arvulisteks väärtusteks, mida seejärel on võimalik töödelda.

Järgmise sammuna saabki võtta Arduino Nano ja ühendada selle mõõtmisteks signaaligeneraatori külge.

Joonis 2. Praktikumis kasutusel olev signaaligeneraator Rigol DG812.

Sooritades praktikumi kaugelt on mõistlik praktikumijuhendajaga kontrollida, et riistvara on korrektselt ühendatud.

Alles peale allpool olevate ühenduste sooritamist tuleb ühendada Arduino arvutiga. Ülesande esimeses osas tuleb ühendus teha järgnevalt:

- Arduino GND → signaalikaabli GND, märgitud joonisel 1-ga;
- Arduino A0 sisend → signaalikaabli otsik, märgitud joonisel 2-ga.

Kasutades varasemaid teadmisi või uurides näiteid Arduino dokumentatsiooni kategooriatest `analog` ja `communication`, tuleb teha Arduino programm, mis sooritab mõõtmise iga 50 ms tagant ja saadab selle üle *serial* ühenduse arvutisse. Programmi töötavust saab kontrollida, kui Arduino

keskkonna ülemisest paremast nurgast avada `serial monitor`. Eduka lahenduse korral peaksid sinna ilmuma täisarvud, mille väärtus jääb vahemikku 0-1023.

Eelnevalt võis näha, et numbrite jada paremaks illustreerimiseks on mõistlik kasutada graafikuid, seega tasub vaadata ka `Serial Plotter` nimelise tööriista väljundit. Selle leiab, kui liikuda menüüs **Tools › Serial Plotter**.

Kuna selles aines on signaali vaja ka töödelda, tuleb saadud info vastu võtta Pythonis kirjutatud programmi abil. Selleks tuleb esimese sammuna uurida Pythoni *serial* teegi dokumentatsiooni ja aluskoodi `yl4.py`. Kasutades seda koostada programm, mis teeb järgnevat:

1. kõigepealt loob õigete parameetritega *serial* ühenduse Arduinoga;
2. *QTimeri* poolt kutsutavas funktsioonis kontrollitakse, kas arvuti *serial* puhvris on saadaval uut infot.
    a. Kui *serial* puhvris on infot, siis loetakse kogu saadaolev info ridahaaval sisse.
3. Viimased 500 kogutud andmepunkti kuvatakse PyQtGraph graafikul.

Kui x-teljel on täisarvulised indeksid, siis pole `plot` funktsiooni väljakutsel graafikule x-telje väärtusi vaja anda, piisab ainult väärtuste argumendist.

Ülesande arvestatuks saamiseks tuleb teil:

1. demonstreerida loodud Arduino koodi;
2. demonstreerida loodud Python koodi;
3. koguda andmeid vähemalt 10 sekundit ning salvestada need kasutades NumPy't.
    a. Salvestatud andmeid läheb vaja koduülesande lahendamisel.

Näidake oma lahendust juhendajale ning tehke commit!

Kodus tuleb lahendada Kolmas koduülesanne

1. Uurige signaali, mille salvestasite neljandas praktikumiülesandes. Signaal, mida selles ülesandes mõõtsite, on morse. Kirjeldage kodutöös reegleid, kuidas signaal morses kodeeritakse ja seejärel dekodeerige antud signaaliga edastatud sõnum.

    Eeldatav vastus:

    - kirjeldab lühidalt probleemile lähenemist ja dekodeeritud sõnumit.

3. Lisaülesanded (Valikuline)

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

5. ülesanne: Morsedekooder (0.5 punkti)

Selle ülesande lahendamiseks tuleb koostada programm, mis sarnaselt 4. ülesandega loeb sisse Arduinost saadava pinge, aga lisaks selle graafikul kuvamisele dekodeerib seda jooksvalt morsekoodiks. Programmi väljundina tuleb jooksvalt kuvada ekraanil Arduinost saadavat signaali, omal valikul kas ladina tähestiku või morses kasutatavate sümbolite kujul.

Näidake oma lahendust juhendajale ning tehke commit!

4. Kodutöö ülesanded

1. ülesanne

1. Lisage oma programmi võimalus mõõta aega, kasutades aluskoodis demonstreeritud timeit funktsionaalsust. Võrrelge implementeeritud valemi naiivse ja vektoriseeritud kuju töökiirust erinevate sisendite korral ning kujutage tulemusi tabelis.

   Eeldatav vastus:

   - sisaldab vähemalt viie erineva sisendiga sooritatud mõõtmiste tulemusi, kusjuures vähima ja suurima sisendi vahe peab olema vähemalt tuhat korda;
   - võrdleb ning kirjeldab saadud tulemusi lühidalt.

2. Uurige NumPy poolt pakutavaid võimalusi ning kirjutage vähemalt ühest funktsioonist, mis võiks teid tulevikus aidata signaalide uurimisel.

   Eeldatav vastus:

   - kirjeldab lühidalt, mida valitud funktsioon teeb ning miks te just selle valisite;
   - sisaldab linki vastava funktsiooni dokumentatsioonile NumPy koduleheküljel;
   - demonstreerib paari koodireaga funktsiooni tööd.

2. ülesanne

1. Uurige oma loodud graafikuid. Mis on nendel graafikutel olevate siinus- ja koosinusfunktsioonide perioodid ja sagedused? Millised oleksid periood ja sagedus, kui telgedel kujutatud ühikud ei oleks mitte sekundid, vaid minutid?

   Eeldatav vastus:

   - kirjeldab lühidalt probleemile lähenemist ja arvutuskäiku;
   - toob eraldi välja arvutuskäigus kasutatud valemid.

2. Otsige PyQtGraph'i dokumentatsioonist veel vähemalt 2 erinevasisulist funktsiooni, mis tunduvad vajalikud ja millega saab muuta graafiku teatud parameetreid. Proovige neid funktsioone programmis kasutada ning lisage tulemustest pildid aruandesse. Kirjeldage aruandes oma sõnadega, mida need funktsioonid teevad ja tooge välja olukorrad, kus need kasulikuks osutuksid, koos põhjendusega.

   Eeldatav vastus:

   - kirjeldab lühidalt, mida valitud funktsioon teeb ning miks te just selle valisite;
   - sisaldab linki vastava funktsiooni dokumentatsioonile PyQtGraph koduleheküljel;
   - demonstreerib paari koodireaga funktsiooni tööd;
   - sisaldab pilti funktsiooni rakendamise tulemusest;
   - orignaalmõõdus pildid demonstreeritavatest graafikutest on lisatud teie repositooriumi.

3. ülesanne

1. Uurige signaali, mille salvestasite neljandas praktikumiülesandes. Signaal, mida selles ülesandes mõõtsite, on morse. Kirjeldage kodutöös reegleid, kuidas signaal morses kodeeritakse ja seejärel dekodeerige antud signaaliga edastatud sõnum.

   Eeldatav vastus:

   - kirjeldab lühidalt probleemile lähenemist ja dekodeeritud sõnumit.