
PHYS114



Grunnleggende målevitenskap og eksperimentalfysikk

Laboratorieoppgave 3

PC-basert datainnsamling og simulering

Institutt for fysikk og teknologi
Det matematisk-naturvitenskapelige fakultet

1 Dokumentkontroll

Tabell 1: Versjonshistorikk

| Versjon | Dato | Ansvarlig | Kommentar |
|---------|----------|-------------------|--|
| r1.0 | 29.04.20 | johan.alme@uib.no | Førsteutkast av oppgave 3 med bruk av Python. Tidligere har labview/matlab vært benyttet for det samme, og selve oppgaven har ikke endret seg nevneverdig i forhold til tidligere. |
| r1.1 | 06.08.20 | johan.alme@uib.no | Oppgaven er ferdigstilt. Mangler kun kapittel 4 og korrekturlesing. |
| r1.2 | 31.01.23 | johan.alme@uib.no | Ryddet i oppgaven og reduserte omfanget. Oppdaterte kodeeksemplene. |
| | | | |
| | | | |
| | | | |
| | | | |

2 Målsetting med oppgaven

Målsettingen med laboratorieoppgaven er å oppnå grunnleggende forståelse og erfaring med PC-basert datainnsamling samt bruk av PCen som et simulerings-verktøy.

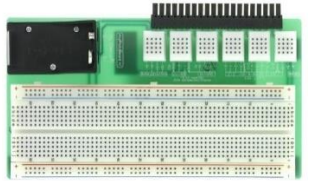



I laboratorieoppgaven skal det utvikles et termometer basert på en termistor. Videre skal Nyquists samplingsteorem studeres sammen med Fourier-analyse av komplekse signal. Avslutningsvis i laboratorieoppgaven ser vi på Monte Carlo simulering. Dette bruker tilfeldige tall generert i PCen i sammenheng med en modell for å simulere oppførsel.

I laboratorieoppgaven benyttes programmeringsspråket Python, enten ved å bruke Visual Studio Code eller å bruke programmeringsmiljøet Anaconda/Spyder.

3 Laboratorieutstyr benyttet i oppgaven

En komplett utstyrsliste for laboratorieoppgave 3 er gitt i Tabell 2.

Tabell 2: Utstyrsliste, laboratorieoppgave 3

| Type | Merke | Illustrasjon |
|---------------------------------|---|--|
| PC med Python utviklingsverktøy | Visual Studio Code eller Anaconda/Spyder (vist i bilde) |  |
| NI myDAQ datainnsamlingsenhet | National Instruments |  |
| myProtoBoard | National Instruments |  |
| BNC adapter board | National Instruments |  |
| 10 kΩ motstand | - | |
| NTC termistor | Farnell 730-051 |  |
| Diverse ledninger | - | |
| Mikrofon | Shure SM-58 |  |
| Digitalt multimeter | Keithley 2100 |  |
| Signalgenerator | Agilent 33220A |  |
| Oscilloskop | Tektronix DPO2002B |  |
| Isbiter og beholder | - | |

4 Oppgave

NB! Rapporten skal inneholde all kode, utdrag fra alle CSV-filer og alle plott som blir etterspurt i oppgaven.

4.1 Del 1 – Måling av temperatur med termistor

4.1.1 Termistoren

Det finnes mange ulike typer sensorer for måling av temperatur. Termistoren, som er en av disse, er en elektronisk komponent som endrer elektrisk motstand ved endring i temperatur.

Termistoren vi skal bruke i denne oppgaven er en såkalt NTC-termistor av merke Farnell 730-051. NTC betyr «Negative Temperature Coefficient», som vil si at motstandsverdien avtar når temperaturen øker.

Sammenhengen mellom motstandsverdi R_θ og temperatur θ er gitt som:

$$R_\theta = K e^{\frac{\beta}{\theta}} \quad (1)$$

R_θ avtar altså eksponentielt med θ . Temperaturen θ er oppgitt i Kelvin slik at temperatur i grader Celsius er gitt som

$$T[^\circ\text{C}] = \theta - 273.15 \quad (2)$$

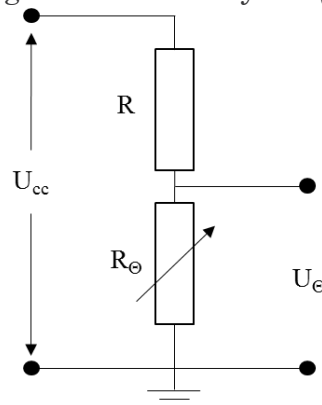
Denne termistoren (*Farnell 730-051*) er laget slik at motstandsverdien ved 25°C er lik $10\text{ k}\Omega$. I ligningen (1) ovenfor er K og β konstanter for den aktuelle termistortypen. Disse kan finnes ut fra to punkter i termistorens datablad:

Tabell 3: Utvalgte data fra databladet til termistor Farnell 730-051

| $T[^\circ\text{C}]$ | 0 | 10 | 25 | 50 | 100 |
|---------------------|-------|-------|-------|------|-----|
| $R_\theta [\Omega]$ | 32063 | 19679 | 10000 | 3667 | 712 |

Siden ligning (1) ovenfor er en tilpasning, vil konstantenes verdier være avhengige av hvilke punkter som velges. I denne oppgaven skal vi lage et elektronisk termometer med et måleområde fra 0°C til 50°C .

4.1.2 Bruk av datainnsamlingsenheten NI myDAQ



Figur 1: Spenningsdeler for konvertering fra en målt motstandsverdi (R_θ) til en spenning (U_θ) som videre kan digitaliseres.

Inngangene på *NI myDAQ* kan kun måle elektrisk likespenning, ikke motstandsverdier. Vi må derfor lage en enkel krets på *myProtoBoard* som konverterer motstandsverdien tilhørende termistoren til en elektrisk spenning. Deretter kan vi lese disse spenningsverdiene inn til datamaskinen via datainnsamlingsenheten.

Den enkleste måten å gjøre dette på er ved å benytte en spenningsdeler-krets, som vist i Figur 1. Spenningsdeleren er basert på at drivspenningen (U_{CC}) og den øverste motstanden (R) er konstante slik at en forandring i termistorens motstand (R_θ) medfører at spenningsforholdet over R og R_θ forandres. I denne oppgaven bruker vi $U_{CC} \approx 5,0 \text{ V}$ og $R \approx 10 \text{ k}\Omega$.

4.1.3 Laboratorieutstyr

For å gjennomføre denne deloppgaven skal følgende instrumenter, komponenter og programvare brukes:

- Termistor *Farnell 730-051*
- $10 \text{ k}\Omega$ resistor
- Multimeter *Keithley 2100*
- *NI myDAQ* datainnsamlingsenhet
- *myProtoBoard* tilkoblingsmodul for *NI myDAQ*
- Isbiter
- Beholder for isbiter
- Programvare for programmering i Python

4.1.4 Gjennomføring

Del 1.1: Termistoren: Beregning, plotting og tapsfaktor

1. Vis at $K = 0.0263 \text{ }\Omega$ og at $\beta = 3828 \text{ Kelvin}$ på grunnlag av termistorverdiene ved 0°C og 50°C gitt i Tabell 3.
2. Bruk Python biblioteket *matplotlib* og plott funksjonen gitt i likning (1) i temperaturområdet $0 - 50^\circ\text{C}$. Husk benevnning på aksene og tittel på plottet. For å regne ut R_θ må dere bruke biblioteket *Numpy*.¹

Del 1-2: Datainnsamling ved hjelp av måleinstrument

3. Koble NTC-termistoren til Keithley 2100 multimeteret og still dette inn på å måle motstand. Sjekk at verdien i displayet er som forventet i henhold til romtemperatur og plottet dere lagde i oppgave 3a-1.2. Termistoren bør ha en verdi rundt $10 \text{ k}\Omega$.

Keithley 2100 multimeteret er koblet til PCen på labplassen med en USB-kabel og det er mulig å kommunisere med instrumentet via serieport over USB. For å lese av motstandsverdien benyttes kommandoen **MEAS:RES?**. Denne kommandoen vil lese en motstandsverdi fra multimeteret.

4. Lag et skript *readKeithley.py* basert på eksempelet gitt i Listing 1. Her skal dere ved hjelp av **MEAS:RES?** funksjonen lese en enkelt verdi. Verdien skal *tidsstemples*, altså skal dere lese av datamaskintiden samtidig som dere henter verdien. Dette skal så skrives ut på kommandolinjen. Python biblioteket *pyvisa* skal benyttes til å lese verdien.

¹ Bruk gjerne Python koden fra lab 1 som inspirasjon for å få til dette.

5. Utvid koden i oppgave 4 til å skrive ut temperatur i Celsius på kommandolinjen i stedet for motstandsverdi. Dere må bruke *numpy* til å gjøre beregningen.
6. Utvid koden i oppgave 5 til å lese ut 100 verdier og skrive de til terminalvinduet. Forsøk å vekselvis varme opp og kjøle ned termistoren med å for eksempel holde den inni håndflaten (ikke bruk lighter!).

```
# Import av bibliotek for å snakke med Keithley multimeter
import pyvisa as pv
from datetime import datetime

# Lister hvilke ressurser som er tilgjengelig for å kommunisere med
# og skriver ut listen på kommandolinjen
rm = pv.ResourceManager()
l = rm.list_resources()
print(l)
# Basert på printen så ser man at Keithley multimeteret er første enhet i listen
# (Eneste USB instrumentet) NB! Dette kan endres dersom flere instrument blir skrudd på
keithley = rm.open_resource(l[0]) # Velger første element i listen av instrumenter

# leser tidsstempel
timestamp = datetime.now()
# leser en motstandsverdi og konverterer til flyttall
value = float(keithley.query('MEAS:RES?'))
# skriver den ut på kommandolinjen
print("%s - Termistor motstand: %5.2f Ohm" %(timestamp.time(), value))
rm.close() # Lukker pyvisa ressursen
```

Listing 1: Skript for å lese en enkelt verd fra Keithley multimeter

Del 1-3: Datainnsamling ved hjelp av NI myDAQ

7. Finn et uttrykk for spenningen U_θ som skal leses med datainnsamlingsenheten *myDAQ* som funksjon av R_θ .
Tips: Bruk Ohms lov til å bestemme strømmen gjennom begge motstandene når det antas at motstanden til inngangen på NI *myDAQ* er så stor at strømmen som går inn der er neglisjerbar. Deretter kan Ohms lov brukes enda en gang til å bestemme U_θ .
8. Kombiner uttrykkene du fant i oppgave 5 og oppgave 7 og vis at temperaturen oppgitt i °C kan uttrykkes som:

$$T[^\circ\text{C}] = \frac{\beta}{\ln \left[\frac{U_\theta R}{K(U_{cc} - U_\theta)} \right]} - 273.15 \quad (4)$$

Dette er selve målefunksjonen hvor alt på høyre side er kjent (konstanter), unntatt U_θ som er målevariabelen.

9. Kretsen i Figur 1 skal kobles opp på *myProtoBoard* med U_θ knyttet til *myDAQ* inngang *Ai0*. Før kretsen kobles opp sjekk de faktiske verdiene for $U_{cc} \approx 5.00\text{V}$ og $R \approx 10\text{k}\Omega$ med et multimeter. Bruk de målte verdiene når du senere implementerer målefunksjonen i Python. Det er ikke nødvendig å beregne usikkerhet av U_{cc} eller R i denne oppgaven. Gjør likevel et par tre målinger av hver parameter for å sikre at feilmåling ikke blir gjort.

Hvis termistoren brukes til å måle i et medium med dårlig varmeledningsevne, f.eks. luft, må det tas hensyn til at den elektriske strømmen gjennom termistoren medfører en liten egenoppvarming. Dette fører videre til at avlest motstandsverdi blir litt for liten. Ifølge

termistorens datablad oppgis at det i luft skal ca. 2.6 mW til for å gi en feilavlesning på 1°C ? (Den såkalte tapsfaktoren i luft er $\approx 2.6 \text{ mW}/^\circ\text{C}$).

10. Hvor mye utgjør denne feilavlesningen i $^\circ\text{C}$ ved romtemperatur (ca 25°C)?

Er det mulig å korrigere/reducere denne feilavlesningen?

Hint: Den elektriske effekten i termistoren er gitt ved sammenhengen $P = UI$, hvor U er spenningen over termistoren og I er strømmen gjennom termistoren.

11. Test kretsen ved å lese ut en enkelt verdi fra myDAQ. Dette kan gjøres med scriptet gitt i Listing 2. Det er forventet at dere skal lese en verdi rundt 2.5 V siden vi har termistoren i romtemperatur.

```
# Importerer nødvendige biblioteker
import nidaqmx
from datetime import datetime

# Oppretter konstanter for kommunikasjon med myDAQ
MYDAQ_NAME = 'myDAQ2'
CH_NAME = 'ai0'

# Oppretter en task og forteller at Tasken skal lese inputspenning fra
# myDAQ og kanaler gitt av konstantene
get_ai0_V = nidaqmx.Task()
get_ai0_V.ai_channels.add_ai_voltage_chan(physical_channel=MYDAQ_NAME+ "/" + CH_NAME)

# Tidsstempler målingen
timestamp = datetime.now()
# read funksjonen returnerer en liste med 1 verdi siden vi kun leser 1 sample
# Vi er dermed kun interessert i verdien på index 0 i listen
value = get_ai0_V.read(number_of_samples_per_channel=1)[0]
# Vi skriver den ut på kommandolinjen med tidsstempel
print("%s - Spenning U_theta: %5.2f Volts" %(timestamp.time(), value))
# Avslutter med å lukke myDAQ ressursen
get_ai0_V.close()
```

Listing 2: Kodelinjer som behøves for å lese en verdi fra myDAQ.

Nå skal vi lese ut verdier kontinuerlig og samtidig plotte de *live* mens vi leser. For å gjøre dette skal vi lage to script som skal kjøre samtidig. Det første skriptet skal lese data og skrive det til en CSV fil. Det andre skriptet vil lese CSV filen og lage plott av dataene.

12. Lag et script *read_voltage.py* som leser spenningen fra *myDAQ*, legger til et tidsstempel og skriver dette til en CSV-fil. Et skjelettprogram til dette skriptet er angitt i Listing 3. Fyll inn kode for å gjøre dette mellom linjene «KODE SKRIVES HER» og «KODE SLUTT». Dokumenter i rapporten hva resten av koden gjør slik at dere viser at dere forstår hva som skjer.

13. Lag et skript *liveplot_temp.py* som plotter temperaturen i $^\circ\text{C}$ som funksjon av tiden. Dette skriptet skal:

- Kontinuerlig lese CSV-filen.
- Beregne temperatur fra spenningsverdien. Bruk formel (4) og de målte verdiene av R og U_θ til dette.
- Plotte verdiene i et plott med tid på x-aksen og temperatur på y-aksen.

Prøv å varme opp termistoren i hånden for å se endring i temperatur.

```

# -*- coding: utf-8 -*-
import nidaqmx
import csv
from datetime import datetime
import threading as th

# Filnavn
FILENAME = 'data_mydaq.csv'

# Oppretter konstanter for kommunikasjon med myDAQ
MYDAQ_NAME = 'myDAQ2'
CH_NAME = 'ai0'

# Global variabel som er sann inntil vi trykker <enter>
keep_going = True

# funksjon som lytter etter tastetrykk, og hvis dette skjer så avsluttes scriptet
def key_capture_thread():
    global keep_going
    input("\n Avslutt programmet med å trykke <enter>\n")
    keep_going = False

# Funksjon som leser spenningsverdien fra NI myDAQ. Denne leser verdier kontinuerlig
# inntil variabel keep_going blir usann (mao når enter blir trykket).
# Dataene formateres i en tuple med tidsstempel og avlest verdi dette blir skrevet til
# en CSV (comma separated value) fil.
# myDAQ-driver blir lukket når man avslutter programmet
def read_voltage():
    # Egen tråd som lytter etter tastetrykk blir startet
    th.Thread(target=key_capture_thread, args=(), name='key_capture_thread',
              daemon=True).start()

    # ---KODE SKRIVES HER
    # initialiser myDAQ og velg å koble til mydaq og kanal
    # ---KODE SLUTT

    # Oppretter CSV fil med to felt: 'timestamp' og 'voltage'
    # Skriver Header og lukker filen ('with'-konstruksjonen sikrer trygg lukking)
    fieldnames = ["timestamp", "voltage"]
    with open(FILENAME, 'w') as csv_file:
        csv_writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        csv_writer.writeheader()

    # Her leses data og skrives til CSV-fil
    # Vi skal lese en verdi om gangen og skrive denne til fil.
    # Koden skal avslutte når keep_going er false
    while keep_going:
        with open(FILENAME, 'a') as csv_file:

            # ---KODE SKRIVES HER
            # opprett en tuple som heter datarow, der første element er tidsstempel
            # og andre element er spenningsverdi. Skriv denne til CSV fil.
            # ---KODE SLUTT
            print(datarow) # printer verdiene til terminalen

    get_ai0_V.close()

# Her kjøres funksjonen read_voltage(). Dette er hele programmet.
read_voltage()

```

Listing 3: Skript for å kontinuerlig lese verdier fra myDAQ og skrive de til en CSV-fil


```

# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from datetime import datetime

# Hvilken fil leses?
FILENAME = 'data_mydaq.csv'

# Definisjoner av konstanter
# !!Bruk målte verdier for R og UCC!!
BETA = 3828
K = 0.0263
R = 10000
UCC = 5

# Formatet til tidsstemplene i CSV-filen:
# År-Måned-Dag Time:Minutt:Sekund.Millisekund
dt_format = '%Y-%m-%d %H:%M:%S.%f'

# Dette er ikke nødvendig, men gir penere plot - velg deres eget design hvis dere vil
font = {'family': 'serif',
        'color': 'black',
        'weight': 'normal',
        'size': 10,
        }

# https://matplotlib.org/stable/gallery/style\_sheets/style\_sheets\_reference.html
plt.style.use('fivethirtyeight')

# Funksjon som henter data fra CSV fil og plotter de
def animate(i):
    # Henter ut data fra filen
    data = pd.read_csv(FILENAME)
    # Leser dato som tekst og konverterer til datetime objekter
    x = list(map(lambda x: datetime.strptime(x, dt_format), data['timestamp']))

    # --- SKRIV KODE HER
    # Les spenning fra fil og regn ut temperatur
    # --- KODE SLUTT

    # Sletter plott - og sikrer automatisk oppdatering av x-aksen
    plt.cla()
    plt.gcf().autofmt_xdate()
    # --- SKRIV KODE HER
    # Plotting av dataene. Dere skal lage en label, og navngi x og y akse
    # for å bruke stylesheet gitt av font i x og y akse, bruk parameter fontdict=font
    # --- KODE SLUTT

# Dette sikrer at vi har "live"-plotting ved hjelp av FuncAnimation biblioteket
ani = FuncAnimation(plt.gcf(), animate, interval=100)
plt.tight_layout()
plt.show()

```

Listing 4: Skript for å plotte data «live» fra en CSV-fil

I siste delen av oppgaven skal dere legge termistoren i en beholder med isvann og vente en stund til temperaturen stabiliserer seg. Vi kan da anta at blandingen har temperatur omtrentlig lik $0\text{ }^{\circ}\text{C}$ så lenge det er is igjen i blandingen.

14. Ta data slik dere gjorde i oppgave 12 og 13. Følg med på temperaturen over tid ved å se på liveplottet.

- i) Når temperaturen er noenlunde stabil, kan du starte datatagningen ved å starte `read_voltage.py` på nytt. La denne kjøre i ca 2-3 minutter for å samle inn nok data til å få akseptabel statistikk².
- ii) Basert på dataene som er lagret i CSV-filen skal dere lage et script som plottet et histogram av dataene. Pass på å få korrekt *bin*-størrelse³ i histogrammet. Programmet skal også beregne middelerdi og standardavvik og markere disse i plottet. Se Listing 5.

For å beregne standardavvik og middelerdi kan funksjonene *mean* og *std* fra biblioteket *numpy* brukes. Funksjonen *std* må ta inn en parameter *ddof=1* slik at standardavviket basert på et utvalg blir beregnet, som gitt i formel (5).

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (5)$$

For å lage et histogram må funksjonen *hist* brukes fra biblioteket *matplotlib.pyplot*. Fra samme bibliotek kan funksjonen *axvline* benyttes for å tegne en vertikal linje i plottet for å markere middelerdi og standardavvik.

² Hvis dataene drifter i deler av måleserien, kan dette fjernes etterpå slik at histogrammet kan lages kun av dataene der temperaturen er stabil.

³ Bin-størrelse er spennet i måleverdier. F eks for lab 1 (sterk kilde) så hadde vi en inndeling lik 1000-1009, 1010-1019 osv. Bin-størrelsen i dette tilfellet var altså 10. Dere kan gjerne ta utgangspunkt i koden fra lab1 for å lage dette histogrammet.

```

# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# KONSTANTER
BETA = 3828
K = 0.0263
R = 10000
UCC = 5

#FILNAVN
FILENAME = "data_mydaq_histogram.csv"
# Setter størrelse på bin
BINSIZE = 0.01

# Stil for tekst i plot - ikke nødvendig, men plottet blir penere
font_axis = {'family': 'serif',
              'color': 'black',
              'weight': 'normal',
              'size': 12,
              }
font_lbl = {'family': 'serif',
            'color': 'black',
            'weight': 'normal',
            'size': 10,
            }
# Stylesheet for plot
plt.style.use('fivethirtyeight')

# Funksjon for å lage bins ved å justere BINSIZE
def bins(Temperature):
    min_value = Temperature.min()
    max_value = Temperature.max()
    low_limit = min_value - 0.1
    high_limit = max_value + 0.1
    bins = np.arange(start=low_limit, stop=high_limit, step=BINSIZE)
    return bins

# Leser inn fil
data = pd.read_csv(FILENAME)
# leser spenningsverdier. Hvis man ønsker å ta et subset av samplene kan man man bruke
# U_theta = data['voltage'].loc[0:999] # Denne leser de først 1000 samplene
U_theta = data['voltage']

# ---KODE SKRIVES HER
# 1. Beregn temperatur
# 2. Beregner middelerdi og standardavvik for å markere i plott
# 3. Plotting av data begynner her - bruke hist funksjon på plt
# 4. Legger inn linjer for middelerdi og standardavvik - men beskrivende label
# --- KODE SLUTT HER

plt.xlabel("Temperatur bins, bin størrelse = %.2f" %BINSIZE, fontdict=font_axis)
plt.ylabel("Counts", fontdict=font_axis)
plt.gca()
plt.tight_layout()
plt.show()

```

Listing 5: Skript for å lage histogram av temperaturdata

4.2 Del 2 - Sampling av tidskontinuerlige, periodiske signal

4.2.1 Bakgrunn

I denne oppgaven skal vi studere hurtig varierende signaler og nødvendigheten av å oppfylle Nyquists sampleteorem⁴. Moderne måle- og reguleringsystemer inneholder nesten uten unntak en digital enhet som gjør det nødvendig å omforme signaler mellom analog og digital form; signalet må derfor *digitaliseres*.

Dette betyr at to ting må gjøres med det kontinuerlige *analoge* målesignalet;

- *Sampling* av målesignalet med faste mellomrom slik at dets verdier bare foreligger ved *diskrete* tidspunkter.
- *Kvantisering* av signalets verdier ved hver sampling.

Utfordringen med digitalisering er at informasjon kan gå tapt. Dette betyr i praksis at samplingen må gjøres ofte nok og kvantiseringen må ha tilstrekkelig god oppløsning. Sampler vi derimot for ofte og med for god kvantisering genereres unødvendig store datamengder, noe som kan være en utfordring, spesielt ved *strømming* av data.

Det som er viktig er at det opprinnelige kontinuerlige signalet skal kunne rekonstrueres på grunnlag av det digitaliserte signalet. Tiden mellom hver kvantisering kalles samplingstiden, ΔT_s . Hvis ΔT_s er kort nok slik at samplingen gjøres ofte i forhold til variasjonene i det analoge signalet, så vil digitaliseringen ikke gi noe tap av informasjon.

Nyquists sampleteorem angir hvor ofte samplingen må gjøres for at dette skal være oppfylt:

“Et båndbreddebegrenset, kontinuerlig signal kan bli samlet og perfekt rekonstruert fra samplepunktene dersom signalet er samlet dobbelt så raskt som den høyeste frekvenskomponent i signalet”:

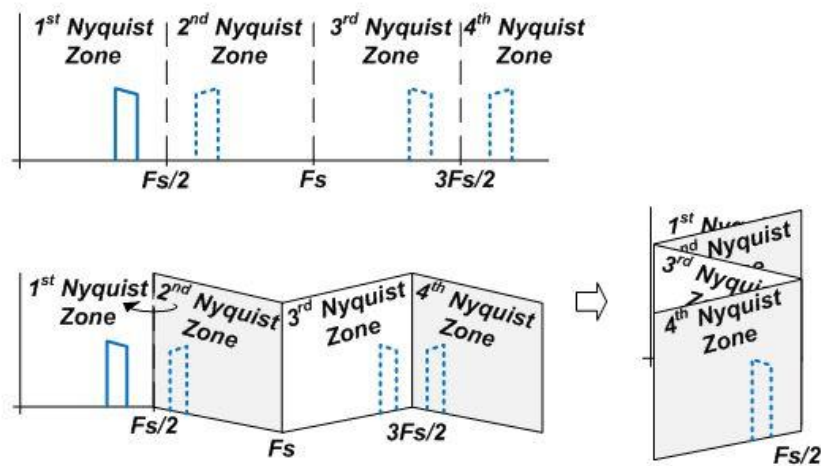
$$f_s = \frac{1}{\Delta T_s} \geq 2f_{maks}$$

hvor f_s er samplingsfrekvensen og f_{maks} er den høyeste frekvenskomponenten til det analoge signalet⁵. Hvis samplingstiden blir for lang i forhold til signalperioden er ikke Nyquists samplingsteorem oppfylt. Da kan ikke det opprinnelige signalet rekonstrueres, og den rekonstruerte signalfrekvensen har en lavere frekvens enn hva det opprinnelige signalet har. Denne effekten kalles *nedfolding* (*aliasing*).

Begrepet nedfolding brukes siden signalene “folder” seg rundt samplingspunktene, slik som illustrert i Figur 2. Ved *nedfolding* vil frekvensen på signalet vi rekonstruerer i absoluttverdi være lik differansen mellom samplingsfrekvensen f_s og den originale signalfrekvensen f . Dette betyr for eksempel at hvis et signal med signalfrekvens $f = 1000 \text{ Hz}$ samples med samplingsfrekvens $f_s = 1200 \text{ Hz}$ så vil frekvensen på det rekonstruerte signalet være $f_R = 200 \text{ Hz}$.

⁴ Bakgrunnsstoff for denne oppgaven finnes i kapitlet «digitalisering» i veiledningsdokumentet.

⁵ I praksis er f_{maks} den høyeste frekvenskomponenten som ikke kan neglisjeres med hensyn på bidrag til signalets totale effekt.



Figur 2: Illustrasjon av nedfolding. Figur hentet fra Texas Instruments⁶

Sammenhengen mellom antall samplinger, samplingsfrekvens og totalt tidsintervall som samples er gitt ved formel 6.

$$n = f_s T_{tot} = \frac{T_{tot}}{\Delta T_s} \quad (6)$$

der n er antall samplinger, f_s er samplingsfrekvens, ΔT_s er tiden mellom hver sampling (samplingstiden) og T_{tot} er totalt tidsintervall som samples.

4.2.2 Laboratorieutstyr

For å gjennomføre denne deloppgaven skal følgende instrumenter, komponenter og programvare brukes:

- Signalgenerator Agilent 33220A
- Oscilloskop Tektronix DPO2002B
- NI myDAQ datainnsamlingsenhet
- BNC adapter board tilkoblingsmodul for NI myDAQ
- Programvare for å programmere i Python

4.2.3 Gjennomføring

1. I denne oppgaven skal BNC adapter board brukes på myDAQen. Kople signalgeneratoren Agilent 33220A til analog inngang 0 (ai0) på myDAQ og samtidig til et oscilloskop. Signalgeneratoren skal gi ut et sinussignal med frekvens $f = 1000 \text{ Hz}$ og en amplitude $U = 2 V_{p-p}$. V_{p-p} betyr $V_{peak-to-peak}$. Oscilloskopet brukes til å verifisere at signalgeneratoren gir riktig signal ut. Husk å bruke dempeledd på enden av koaksialkabelen for å forhindre refleksjoner i kabelen.
2. Vi skal lage et nytt skript «read_and_plot.py» som leser en sekvens med 500 samples over et tidsintervall på $T_{tot} = 5 \text{ ms}$.
 - Hva blir samplingsfrekvensen f_s ? Hint: se likning (6)
 «read_and_plot.py» skal plote plote samplingene som funksjon av tiden. Her er det ikke nødvendig å plote det «live», noe som gjør kodingen enklere. Husk tittel på plott og på alle akser. Vis omtrent 5 perioder av signalet i plottet.

⁶ https://e2e.ti.com/blogs_/b/analogwire/posts/rf-sampling-aliasing-can-be-your-friend

Finn signalfrekvensen ved å lese av plottet⁷.

- Er frekvensen som forventet?
3. Sett så samplefrekvensen til $f_s = 1100 \text{ Hz}$ med signalfrekvensen fortsatt innstilt på $f = 1000 \text{ Hz}$. NB! Nå må dere lese lenger tid for å kunne plote 5 perioder.
 - Hvilken frekvens blir nå målt av «read_and_plot.py» skriptet? Er resultatet som forventet?
 4. Sett samplefrekvensen lik signalfrekvensen. Du skal nå fange opp et signal som ser ut til å endre seg mye langsommere med tiden enn signalet fra signalgeneratoren.
 - Hvilken frekvens har nå signalet? Er resultatet som forventet?

Hint: Siden frekvensen vil være svært lav i dette tilfellet er det en fordel å øke antall samples i datainnsamlingen for å kunne bestemme frekvensen. Dere må kanskje sette den til flere minutter. For å få til dette er det mulig å endre timeout-verdien for myDAQen. Standardverdi er 10 sekunder. Prøv å endre denne parameteren til en lengre timeout-verdi, eventuelt sette «Timeout(s)» til «-1», som betyr at systemet venter uendelig lenge, altså helt til datainnsamlingen er ferdig. Dere må gjerne ta data i 2-3 minutter eller lenger for å kunne se en hel periode.

I rapporten skal dere diskutere resultatene oppnådd i deloppgaven sett i lys av Nyquists samplingsteorem.

For programmeringen henvises det for denne oppgaven til video lagt ut på mittuib som forklarer hvordan dette skal gjøres.

⁷ Finn periodetiden T ved å lese av avstanden mellom f.eks., to bølgetopper og finn frekvensen ved å bruke formelen $f = \frac{1}{T}$.

4.3 Del 3 – Fourier-transform og Fourier-rekker

4.3.1 Bakgrunn

I forrige deloppgave brukte vi periodiske signal til å illustrere viktigheten av å sørge for at Nyquists samplingsteorem er oppfylt. For komplekse signal og pulser må samplingsteoremet også være oppfylt.

I denne deloppgaven skal vi undersøke sammenhengen mellom analoge signal som funksjon av tid og som funksjon av frekvens. Alle signaler kan løses opp i sinusformete bølger ved Fourier-transformasjon, som ser på signalet som funksjon av frekvens i stedet for som funksjon av tid.

Fourier-transformasjonen gir amplitude og fase på de forskjellige sinusformete bølgene som trengs for å bygge opp signalet. Amplituden er gitt ved det såkalte amplitudespekteret. Hver frekvens har i tillegg en fase, som kan vises i fasespekteret. I denne deloppgaven skal vi bare studere amplitudespekteret.

Vi skal bruke lyd fra en tilkoblet mikrofon som eksempel på et analogsignal som vi ønsker å studere i frekvensplanet. Vi ønsker altså å studere hvilke frekvenser stemmen produserer, og hvilke amplituder de forskjellige frekvensene har. På denne måten kan vi for eksempel studere hvor høy eller lav frekvens vi kan oppnå med stemmen vår.

Maksimal tonehøyde for grunntonen i en mannsstemme er gjerne godt under 400Hz, men stemmen har harmoniske overtoner med høyere frekvens. En kvinnestemme, som er lysere, har naturlig høyere frekvenser. I denne deloppgaven skal det lages forskjellige typer lyd som for eksempel plystring, sang og vanlig tale. For vanlig tale vil frekvensspekteret ha et relativt irregulært utseende, på grunn av at det er mange frekvenskomponenter som er til stede i signalet. For plystring og sang er situasjonen annerledes. Her ser ut som om bare visse frekvenser er til stede. Ser man på plystre- og sang-signalerne som funksjon av tid kan man se at disse signalene nesten er periodiske. Dette blir så reflektert i frekvensspekteret siden periodiske signal kan utvikles i en *Fourier-rekke*.

En *Fourier-rekke* består av en sum av sinusformete bølger ved forskjellige diskrete frekvenser. Den laveste frekvensen kalles *grunnfrekvensen*. De øvrige frekvensene er overtoner eller *harmoniske frekvenser*. De harmoniske frekvensene er en heltall-multippel av grunnfrekvensen. For eksempel er $f_2 = 2000 \text{ Hz}$, $f_3 = 3000 \text{ Hz}$ og $f_7 = 7000 \text{ Hz}$ harmoniske frekvenser av $f_0 = 1000 \text{ Hz}$. Stemmebåndene kan også produsere overtoner som ikke er harmoniske. Disse tonene høres gjerne ikke så «rene og fine» ut. En sinusformet bølge har imidlertid bare en frekvens, og en slik bølge blir derfor gjerne kalt en ren tone, som kan høres litt steril ut. Det er overtonene som gir klangfarge til lyden.

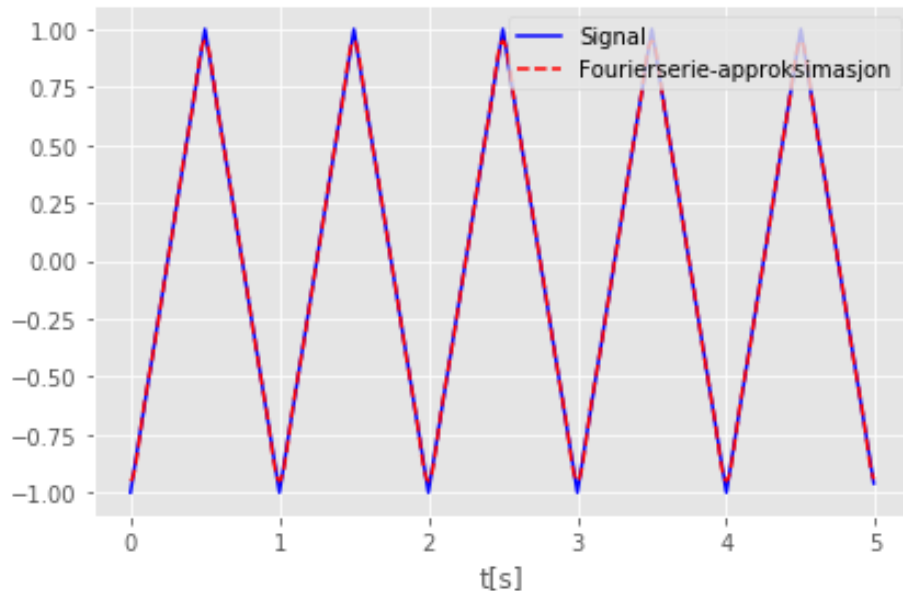
Til slutt i denne deloppgaven skal frekvensspektrene for standard sinus-, trekant- og firkant-signal studeres, og hvordan målte frekvensspektra for disse signalene samsvarer med teorien som gjengitt i Fourier-rekkene for trekant- og firkant-signal.

4.3.2 Fourier-rekker for trekant og firkant signal

Fourier-rekken for et standard trekantsignal, $x(t)$, med en amplitude på 1, er gitt som:

$$x(t) = \frac{8}{\pi^2} \sum_{n=1}^{\infty} \frac{1}{(2n-1)^2} \cos\left(2\pi \frac{(2n-1)}{T} t\right) \quad (7)$$

der T er periodetiden slik at frekvensen til signalet er lik $f = \frac{1}{T}$. Figur 3 viser en Fourierrekke-approximasjon i rød stiplet linje der $T = 1$ sekund og 5 ledd er summert ($n = 1$ til 5). Dette gir et relativt bra trekantsignal sammenlignet med et ideelt trekantsignal som vist i blå heltrukken linje.



Figur 3: Fourierserie-approximasjon med 5 ledd og $T = 1$ sekund.

Hvert ledd i likning (7) er en frekvens i spekteret. Den laveste frekvensen er gitt ved $n = 1$. Dette gir $f_1 = \frac{1}{T}$, den neste frekvensen er ved $n = 2$ som gir $f_2 = \frac{3}{T} = 3f_1$ og så videre. etc. Vi ser at overtonene her er harmoniske, men at de avtar ganske fort i amplitude. Dette er gitt av leddet foran cosinusfunksjonen $\left(\frac{1}{(2n-1)^2}, n = 1, 2, 3, \dots\right)$, altså er amplituden omvendt proporsjonal med kvadratet av den relative frekvensen i forhold til grunnfrekvensen.

Alle periodiske bølgeformer kan uttrykkes som en Fourierserie, der presisjonen av serien er gitt av antall ledd (= antall harmoniske frekvenser) man inkluderer. Dette betyr at tilsvarende formel for en firkantpuls med amplitude 1 er:

$$x(t) = \frac{4}{\pi} \sum_{n=1}^{\infty} \frac{1}{(2n-1)^2} \sin\left(2\pi \frac{(2n-1)}{T} t\right) \quad (8)$$

4.3.3 Laboratorieutstyr

For å gjennomføre denne deloppgaven skal følgende instrumenter, komponenter og programvare brukes:

- Mikrofon
- Signalgenerator *Agilent 33220A*
- *NI myDAQ* datainnsamlingsenhet
- *BNC adapter board* tilkoblingsmodul for *NI myDAQ*
- Programvare for programmering i Python

4.3.4 Gjennomføring

Målinger av lyd med mikrofon

1. Kople mikrofonen til analog inngang 0 på myDAQ-enheten og lag en kode i python som sampler mikrofonsignalet og plottet det som funksjon av tid. Det er nok å ta data i ca 2 sekund. Det anbefales og ta utgangspunkt i koden fra del 2. Samplingsfrekvensen kan settes til $f_s = 8000 \text{ Hz}$.
2. Utvid koden til å lage et plot med 2 subplots der det ene plottet gir signalet som funksjon av tid (oppgave 1), mens det andre plottet viser signalet som funksjon av frekvens, altså et *frekvensspekter*.
Den absolutt enkleste måten å gjøre dette på er via funksjonen *magnitude_spectrum* gitt i *matplotlib*. Det er viktig å sette samplingsfrekvensen riktig slik at korrekt frekvens blir gjengitt på x-aksen. Dessuten, siden vi kun er interessert i de positive frekvensene i signalet⁸, benyttes parameteren *sides='onesided'* for å få dette til.
3. Kjør programmet mens du lager lyd (snakk, plystre, synge) i mikrofonen. Du vil da se en samlet utgave av lyden inn på mikrofonen på i plottene som blir generert. Spenningssignalet vist tidsserieplottet vil være proporsjonalt med lydtrykkbølgen som treffer mikrofonen og dermed vise hvordan denne varierer som funksjon av tiden.
Inkluder minimum 3 plott av signal og frekvensspektra for mikrofonsignal med høye og lave frekvenser i lab-rapporten. Husk å justere plottetiden på signalplottet slik at man lett kan se eventuell periodisitet i signalet.

Erfaringene gjort i denne deloppgaven skal diskuteres i rapporten.

For programmeringen henvises det for denne oppgaven til video lagt ut på mittuib som forklarer hvordan dette skal gjøres.

Målinger med signalgenerator

Kople signalgeneratoren *Agilent 33220A* til analog inngang 0 på myDAQ-enheten, og gjenbruk koden fra målingene med mikrofon. Juster samplingsintervallet til 2 sekunder, men sett samplingsfrekvensen på $f_s = 20 \text{ kHz}$. Plottet av signalet i tidsdomenet skal inkludere 5 perioder av signalet.

I denne oppgaven skal se på en firkantpuls og en trekantpuls og gjennomføre samme målinger for begge signalene. Begge signalene skal ha signalfrekvens $f = 1000 \text{ Hz}$ og amplitude $U = 2V_{p-p}$.

Gjør følgende for begge signalene:

4. Plott signalet i tidsdomenet og plott frekvensspekteret. Inkluder plottene i laboratorierapporten.

⁸ Et signal inneholder både en positiv og en negativ frekvenskomponent siden alle signaler kan uttrykkes ved hjelp av sinusbølger. Fra Eulers formel har man da: $\cos(2\pi ft) = \frac{e^{j2\pi ft} + e^{-j2\pi ft}}{2}$. Den første eksponentialfunksjonen har positiv frekvens, mens den andre har negativ frekvens. Når vi plottet kun den positive halvdelen av signalet, kan vi forvente at kun halvparten av energien i signalet er representert i plottet.

5. Bestem forholdet mellom amplitudene som kommer frem i frekvensspekteret ved å lese av verdien til frekvensene, altså: $\frac{A_{1000}}{A_{1000}}, \frac{A_{3000}}{A_{1000}}, \frac{A_{5000}}{A_{1000}}, \frac{A_{7000}}{A_{1000}}, \frac{A_{9000}}{A_{1000}}$. Stemmer dette forholdet overens med de teoretiske amplitudeforholdene gitt av likning (7) og likning (8)?
Presenter resultatene i en oversiktlig tabell som inkluderer både de målte og de teoretiske amplitude-forholdene.
6. Bruk koden gitt i Listing 6 og plott et trekantsignal.
Modifiser samme kode for å lage et plott av et firkantsignal gitt ved Fourier-rekken i likning (8). Plottet skal vises for $T = 1$ og skal inkludere 5 ledd, altså $n = 1$ til 5. Plott fra $t = 0$ til 5 sekunder.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal

t = np.arange(0,5,0.01) # Tidsvektor (0 til 5 sekunder med steg på 0.01 sekund)
N_MAX = 5               # Antall harmoniske frekvenser
T = 1                   # Periode i sekunder

# Funksjon for å lage Fourierserie
def fourierSeries(n_max,t):
    partialSums = 0
    for n in range(1,n_max): # summerer over antall frekvenskomponenter
        try:
            # Beregner amplitude basert på formel (7) i oppgaven
            bn = 8/(np.pi*(2*n-1))**2
            # Beregner frekvens basert på formel (7) i oppgaven
            wn = 2*np.pi*(2*n-1)/T
            partialSums = partialSums + bn*np.cos(wn*t)
        except:
            print("pass")
            pass
    return partialSums

# lager fourierSerien for trekant signalet (denne koden blir lik for firkant)
f = fourierSeries(N_MAX,t)

# Plotter
plt.style.use("ggplot")
# Bruker signal biblioteket fra scipy for å plotte et trekant-signal
# For å plotte firkant brukes funksjonen signal.square
plt.plot(t,signal.sawtooth(2*np.pi*(1/T)*t-np.pi,width=0.5),color="blue",label="Signal")
plt.plot(t,f,'r--',label="Fourierserie-approksimasjon")
plt.xlabel('t[s]')
plt.title("Fourierserie-approksimasjon med antall ledd = "+str(N_MAX))
plt.legend(loc=1)
plt.tight_layout()
plt.show()
```

Listing 6: Kode for å generere og plotte Figur 3.

Er det samsvar mellom plottene du fikk i spørsmål 4 og de teoretiske plottene du har fått nå?

4.4 Del 4 – Tilfeldige tall og Monte-Carlo simulering

4.4.1 Bakgrunn

Hensikten med denne delen av laboratorieoppgaven er å belyse statistiske begrep ved å bruke en generator av «tilfeldige» tall som er innebygget ethvert programmeringsspråk. Ordet «tilfeldig» står i hermetegn fordi det alltid ligger en algoritme bak genereringen av tilfeldige tall. Når det gjør det, kaller vi prosessen for *pseudotilfeldig*. Vanligvis vil generatoren av tilfeldige tall gi ut et tall som ligger mellom 0 og 1. Disse vil fordele seg helt uniformt på intervallet, og dersom man ikke kjenner algoritmen så vil det se ut som at tallene i sekvensen er helt tilfeldige i forhold til hverandre. Navnet på metoden «Monte-Carlo simulering» kommer fra det berømte kasinoet i bydelen Monte-Carlo i Monaco, der tilfeldige og ikke-deterministiske prosesser avgjør utfallet i roulette og andre hasardspill.

Generatoren for tilfeldige tall i Python er gitt ved funksjonen *random()* i biblioteket *random*.

```
import random
x = random.random()
print(x)
```

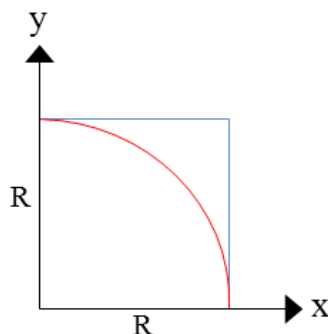
Listing 7: Eksempel på bruk av *random* funksjon i python for å generere ett pseudotilfeldig tall

Eksempel på bruk av *random* funksjonen er gitt i Listing 7. Kjøres dette eksempelet vil et flyttall mellom 0 og 1 genereres. Dette er en *uniform distribusjon*, så alle mulige tall i intervallet er like sannsynlig at blir generert.

De tilfeldige tallene er basert på et *seed* (såkorn på norsk). Dersom ikke dette blir eksplisitt satt til en verdi, benyttes tid siden *epoch*⁹ som *seed*. Dette er tilfellet i eksempelet i Listing 7, og det fører til at et det første tallet som blir trukket ut er forskjellig hver gang programmet kjøres. Hvis *seed* blir satt til en fast verdi, vil man få den samme pseudotilfeldige sekvensen hver gang programmet kjøres, og dermed vil også det første tallet alltid være det samme.

Algoritmen i Python benytter *Mersenne Twister* som generator, og produserer et flyttall med 53 bits presisjon. Perioden, altså antall tall som blir generert før sekvensen gjentar seg selv, er $4.3 \cdot 10^{6002}$ lang.

I laboratorieoppgaven skal den matematiske konstanten π , definert som forholdet mellom omkretsen og diameteren til en sirkel, beregnes med Monte-Carlo simulering.



Figur 4: Kvartsirkel med radius R lagt i et kvadrat med side R .

⁹ epoch er et gitt punkt i tiden som er definert som starttid for datamaskinen. For Unix/Linux er epoch 1. januar 1970. For Windows er det 1. Januar 1601.

For numerisk å beregne π kan det genereres en serie tallpar (x,y) av tilfeldige tall, altså tilfeldig fordelte punkter i et plan. Disse vil fordele seg uniformt i et kvadrat, med nedre venstre hjørne i origo og sidene $R = 1$, som vist i Figur 4. Vi kan slå en kvartsirkel med radius $R = 1$ i dette kvadratet.

Forholdet mellom arealet av sirkelen og arealet av kvadratet er gitt ved:

$$F = \frac{A_S}{A_K} = \frac{\pi R^2/4}{R^2} = \frac{\pi}{4} \quad (9)$$

som er lik forholdet mellom antall punkter med $\sqrt{x^2 + y^2} \leq R$ og det totale antall punkter i kvadratet.

Man trenger selvsagt mange tallpar for å få en god beregning, og formålet med denne oppgaven er blant annet å studere standardavviket av beregningen av π som funksjon av antall genererte tallpar, N .

4.4.2 Teori for presisjonen ved beregning av π

Sannsynligheten for at et tilfeldig tallpar (x,y) skal oppfylle $\sqrt{x^2 + y^2} \leq R$ betegnes med p . Hvis N tallpar genereres, så vil det gjennomsnittlige antallet som oppfyller betingelsen være:

$$\bar{n} = Np \quad (10)$$

I et eksperiment med N tallpar så vil antallet forekomster n som oppfyller betingelsen avvike fra middelveiden og følge en binomisk fordeling. En binomisk fordeling gjelder for eksperimenter som bare har to mulige utfall (her: innenfor eller utenfor).

En kan vise at standardavviket i tallet n er gitt ved

$$\sigma_n = \sqrt{Np(1-p)} \quad (11)$$

I vår beregning har vi at $\pi = 4 \cdot \frac{n}{N}$ slik at vi får at standardavviket σ_π til π blir:

$$\sigma_\pi = 4 \cdot \frac{\sigma_n}{N} = 4 \cdot \sqrt{\frac{p(1-p)}{N}} \quad (12)$$

der p i binomialformelen er sannsynligheten for «treff innenfor» og $(1-p)$ er sannsynligheten for «treff utenfor» kvartsirkelen.

4.4.3 Relevante formler for middelveidi og standardavvik for oppgaven¹⁰

Formlene nedenfor er de teoretiske formlene for de forskjellige størrelsene. Det anbefales å bruke Python og numpy biblioteket til beregningene der det er mulig. Pass da på å bruke funksjonene på riktig måte.

Middelveidi:

¹⁰ Formlene er gjengitt fra veiledningen til laboratorieoppgave 1.

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_N}{N} = \frac{1}{N} \sum_{i=1}^N x_i \quad (13)$$

Standardavvik:

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (14)$$

Usikkerhet i standardavvik:

$$s_s = \frac{s}{\sqrt{2(N-1)}} \quad (15)$$

4.4.4 Laboratorieutstyr

For å gjennomføre denne deloppgaven skal PC med programvare for programmering med Python benyttes.

4.4.5 Gjennomføring

1. Skriv en funksjon

```
def xy_gen(numOfPairs, plot=False, print = False):
```

i python som genererer N tallpar (x,y) , og som bestemmer om tallparene ligger innenfor eller utenfor kvartsirkelen. Totalt antall tallpar N og antall tallpar innenfor kvartsirkelen n skal skrives ut til terminalvinduet.

Funksjonen skal returnere den beregnede verdien til π basert på antall tallpar innenfor kvartsirkelen n og det totale antall tallpar N

Utvid programmet med en ny funksjon

```
def xy_plot(numOfPairs, pi, x, y):
```

som lager et *scatter-plot* over alle tallparene innenfor kvartsirkelen. *numofPairs* er lik N , og x og y er vektorer med x og y koordinatene innenfor kvartsirkelen.

2. Utvid programmet til å kjøre M eksperimenter som hver genererer N tallpar (x,y) og som deretter regner ut:
 - Middelverdien til estimatet av π
 - Standardavviket til estimatet av π
 - Usikkerheten i standardavviket.

Lag et histogram som viser fordelingen av π -estimatene for de M antall forsøk som har blitt gjort.

Kall funksjonen for

```
def pi_trials(numOfTrials, numOfPairs, histogram=False):
```

Antall eksperimenter M skal i resten av lab-oppgaven holdes konstant¹¹, f.eks. $M = 100$.

3. Lag et plott som viser estimert middelværdi av π som funksjon av N med standardavviket inntegnet. Bruk minst 5 verdier av N i figuren, f.eks $N=[100, 1000,$

¹¹ Anbefales likevel å lage en konstant i koden som heter M

10000, 100000, 1000000]. Logaritmisk skala på x-aksen anbefales. Bruk *plt.axhline* i diagrammet for den sanne verdien av π .

4. Lag et diagram med Python som viser standardavviket til estimert π -verdi som funksjon av N med usikkerheten i standardavviket inntegnet. Bruk minst 5 verdier i figuren. I samme plott skal du også plotte det teoretiske standardavviket gitt i seksjon 4.4.2. Bruk igjen logaritmisk skala på x-aksen.

Diskuter resultatene du får.

For programmeringen i denne oppgaven henvises det til videoer på mittuib.

