PHYS114

Grunnleggende målevitenskap og eksperimentalfysikk

Laboratorieoppgave 3

PC-basert datainnsamling og simulering

Veiledning

Institutt for fysikk og teknologi
Det matematisk-naturvitenskapelige fakultet

1 Innhold

2 Dokumentkontroll

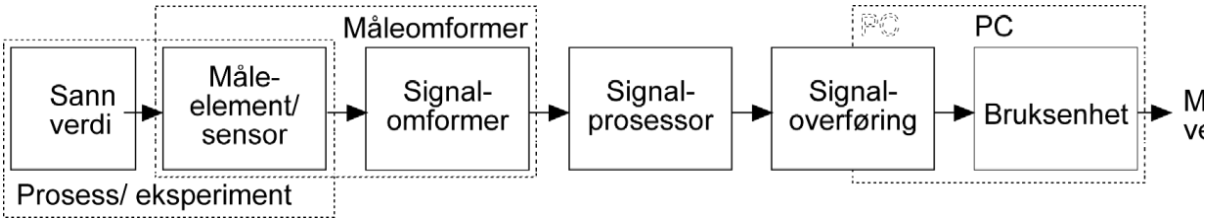
Tabell 1: Versjonshistorikk

Versjon	Dato	Ansvarlig	Kommentar
r1.0	10.08.20	johan.alme@uib.no	Førsteutkast av veiledning til oppgave 3 med bruk av Python. Denne er basert på tidligere veiledning hvor oppgaven skulle gjøres ved labview/matlab.
r1.1	11.01.23	johan.alme@uib.no	Ryddet opp i dokumentet og rettet opp enkelte småfeil. La til informasjon om VScode

3 Målesystemet

Hensikten med et målesystem er å omforme en tilstandsvariabel til en måleverdi som kan presenteres eller brukes på annen måte. Tilstandsvariabelen kan for eksempel være en fysisk eller en kjemisk egenskap. Denne måleverdien representerer da verdien til tilstandsvariabelen med en måleusikkerhet som er bestemt av nøyaktigheten og eventuelle feilkilder til målesystemet og omgivelsene. Sagt på en annen måte er måleresultatet todelt: (1) Man får et estimat av den sanne verdien, og (2) man får informasjon om kvaliteten til estimatet gitt med måleusikkerheten.

For mer utfyllende informasjon om måleusikkerhet og målefeil henvises det til veiledningen til laboratorieoppgave 1 [1].



Figur 1: Skjematisk fremstilling av elementene i et generelt målesystem

Figur 1 viser en generell oppbygning av et målesystem som i sin enkleste form består av et instrument, for eksempel et digitalt termometer. I de fleste tilfeller er de forskjellige elementene delt opp i flere enheter som gjerne kan være fysisk atskilt fra hverandre.

Et typisk målesystem består av følgende elementer:

- **Måleelementet** eller *sensoren* (*sensing element, transducer*) som på en eller annen måte er følsom for den variabelen som skal måles. Termistorer, strekklapper og lysfølsomme dioder er eksempler på måleelementer.
- **Signalomformeren**[1] (*signal conditioning element*) konverterer utgangssignalet fra måleelementet over på en form som er egnet for videre prosessering. Det kan for eksempel være elektrisk spenning eller strøm.
- **Signalprosessoren** (*signal processing element*) behandler og tilpasser utgangssignalet fra signalomformeren i henhold til de krav som stilles av anvendelsen. Dette kan for eksempel være å filtrere signalet, linearisere det eller å konvertere det til digital form. Sistnevnte gjør at en datamaskin kan benyttes som signalprosessor gjennom tilleggsutstyr og/eller programmer.
- **Signaloverføring** (*signal transmission*) i en eller annen form er ofte nødvendig mellom en prosess eller et eksperiment og til et kontrollrom. Kommunikasjon kan foregå på mange ulike måter, som analoge eller digitale signaler over elektriske linjer, fiberoptisk kabel eller trådløst.
- **Bruksenheten** (*signal utilization*) har ofte en rekke funksjoner, alt fra å presentere måleverdien på en hensiktsmessig måte til behandling, prosessering og lagring av måledata. Presentasjonen kan gjøres med enkle indikatorer som lamper og lysdioder, eller paneler med analog og digital visning. Datamaskiner er med sin

[1] Et måleelement og en signalomformer kalles med fellesbenevnelse for en *måleomformer* (transmitter).

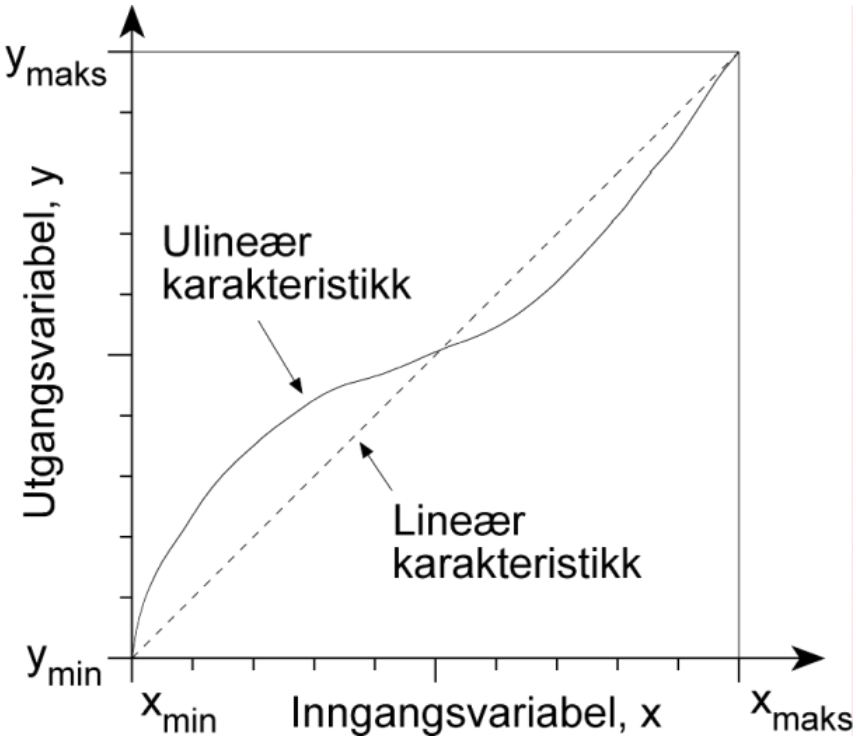
fleksibilitet velegnet til å ivareta alle disse funksjonene. Kontinuerlig lagring av data fra en prosess eller et system omtales forøvrig som datalogging.

Hvilke av disse elementene som rent fysisk er i samme enhet eller boks, varierer. Tilgang på kompakt datakraft gjør at måleelementet, signalomformeren og deler av signalprosessoren ofte er som én enhet plassert i prosessen eller eksperimentet, mens resten er én enhet, for eksempel i et kontrollrom.

De første leddene i et målesystem representerer den største utfordringen. Ideelt sett skal en sensor ha høy følsomhet for den parameteren den skal måle, og være ufølsom for alle andre parametere, for eksempel omgivelsesvariable som f.eks. trykk og temperatur. Siden dette ytterst sjelden er tilfellet, er det viktig å velge optimal sensor, montere og bruke den optimalt, og sørge for at andre parametere påvirker målingen så lite som mulig. Det kan også være nødvendig å måle omgivelsesvariable og ta hensyn til disse når måleresultatet skal beregnes.

3.1 Målesystemets egenskaper

Et målesystem kan karakteriseres ved en rekke grunnleggende parametere som blant annet brukes til å bestemme systemets egenskaper for den tiltenkte anvendelse. Disse fremgår av spesifikasjonene som er oppgitt i *databladet* til sensoren, måleomformeren eller målesystemet.



Figur 2: Eksempler på typiske karakteristikker for et målesystem

Blant funksjonelle egenskaper skilles det mellom systemets statiske og dynamiske karakteristikk. Den statiske karakteristikken beskriver sammenhengen mellom den sanne målevariabelen x (inngangsvariabelen) og måleverdien y (utgangsvariabelen) når målevariabelens verdi er konstant eller kun forandres sakte (*steady state*). Dette er skissert i Figur 2.

Den dynamiske karakteristikken beskriver systemets oppførsel under eller like etter hurtige forandringer i målevariabelens verdi.

Følgende definisjoner er nyttige for statisk karakterisering av et komplett målesystem eller deler av et målesystem, for eksempel måleomformeren:

- **Måleområdet** (*range*). For inngangsvariabelen x er dette intervallet mellom nedre og øvre målegrenser, x_{min} og x_{maks}. For et termometer kan dette være henholdsvis *-10 °C* og *50 °C*. For utgangsvariabelen y er måleområdet det tilsvarende intervallet mellom nedre og øvre grenser, y_{min} og y_{maks}. Hvis utgangsvariabelen er gitt som en elektrisk spenning, kan dette være mellom *0 V* og *5 V*. I tillegg til måleområdet blir det også ofte oppgitt *driftsgrenser* (operative limits) som er nedre og øvre grenser for det området måleomformeren kan benyttes uten å ta varig skade.
- **Linearitet** (*linearity*) angir hvor godt ett plott av y som funksjon av x faller sammen med en rett linje. Kurven som fremkommer kalles systemets karakteristiske kurve eller signatur. Dette kalles også for systemets karakteristikk.
- **Følsomheten** (*sensitivity*) beskriver hvor mye utgangsvariabelen reagerer når inngangsvariabelen endres, det vil si dy/dx i eksempelet i Figur 2.
- **Oppløsningen** (*resolution*) uttrykker et systems evne til å skille mellom nærliggende verdier av inngangsvariabelen.
- **Måleusikkerheten** (*measurement uncertainty*) angir et intervall rundt måleverdien, som er et estimat av den sanne verdien, der den sanne verdien med en viss sannsynlighet ligger. Dessverre er det slik at måleusikkerheten blir oppgitt på mange forskjellige måter. Noen ganger brukes begreper som nøyaktighet og repeterbarhet. Sammenhengen mellom disse blir forklart senere.

I tillegg kommer andre viktige egenskaper som effektforbruk og krav til montering og bruk. Dette kan for eksempel være temperaturområdet eller frekvensområdet målesystemet kan brukes i.

3.2 Måleresultat

Et måleresultat, y, består alltid av en måleverdi som er et estimat av den sanne verdi med en tilhørende måleusikkerhet. Dernest er et måleresultat ofte en funksjon av flere måleverdier x_1, x_2, x_3, …, x_n hver med tilhørende usikkerhet $u(x_1)$, $u(x_2)$, $u(x_3)$, …, $u(x_n)$:

$$y = f(x_1, x_2, x_3, \dots, x_n) \tag{1}$$

Eksempelvis er det ved måling av strømning i rør, ofte også nødvendig å måle trykk og temperatur for å kunne beregne korrekt strømningsrate. I slike tilfeller består målesystemet av én bruksenhet med innganger fra flere måleomformere.

3.3 Måleusikkerhet og målefeil

Den totale måleusikkerheten har opphav i nøyaktigheten til måleinstrumentet, men også miljøpåvirkning som kan gi måleusikkerhet eller til og med målefeil. For å kunne bruke og forstå målesystemer best mulig er det vanlig å klassifisere måleusikkerheten som **type A** eller **type B**.

Type A usikkerhet er usikkerhet fra et beregnet standardavvik på grunn av repeterte målinger. **Type B** usikkerhet er estimert på andre måter enn statistisk databehandling. Usikkerhetene/feilene kan klassifiseres som følger:

- **Statistiske, tilfeldige eller ikke-deterministiske feil**. Dette er feil som gjør at utgangsvariabelen fluktuerer tilfeldig omkring en middelverdi selv om det ikke er noen endringer i inngangsvariabelen. Fluktuasjonene følger en sannsynlighets-fordeling som ofte er normalfordelingen. Denne typen feil omtales helst som støy og kan ofte kvantifiseres, for eksempel gjennom målinger. Årsaken kan blant annet være langsomt varierende fluktuasjoner i signalet som skapes i sensoren eller elektronisk støy i sensorelektronikken. Dette er opphav til **type A** måleusikkerhet.
- **Systematiske feil**. Dette er feil som endrer karakteristikken til et system på en bestemt måte eller i en bestemt retning. Virkningen av systematiske feil er i noen tilfeller kjent, f.eks. fra eksperimentell kartlegging, slik at den kan uttrykkes som funksjon av den aktuelle årsaksvariabelen. *Drift* er at denne typen endrer seg med tiden. Dette bidrar til **type B** måleusikkerhet.
- **Interferens eller deterministiske feil**. Dette skyldes feilkilder hvor påvirkningen på en eller annen måte er forutsigbar. Et godt eksempel er forstyrrelser som skyldes lysnettfrekvensen på *50 Hz*. Dette omtales i mange sammenhenger også bare som støy. Dette bidrar til **type B** måleusikkerhet.
- **Dynamiske eller transiente feil**. Dette er feil som er relatert til systemets dynamiske egenskaper og som oppstår når inngangsvariabelens verdi endres for hurtig i forhold til systems tidskonstant(er). Dette kan bidra både til **type A** og **type B** måleusikkerhet.

3.4 Kilder til målefeil

Det finnes flere feilkilder som det må tas hensyn til når et målesystem skal konstrueres, karakteriseres eller undersøkes ved for eksempel feilsøking. De vanligste feilkildene er:

- **Fluktuasjoner og drift** i omgivelsesvariable (*environmental inputs*) som drivspenning, tetthet og komposisjon av målemediet, trykk, luftfuktighet og temperatur. Fluktuasjoner og forandringer i temperaturen er en vanlig kilde både til statistiske og systematiske feil.
- **Elektromagnetisk støy og interferens** (*EMI - Electromagnetic Interference*). Dette kan gjøre seg gjeldende enten gjennom stråling ved at for eksempel ledningsbaner i målesystemet virker som antenner, eller direkte gjennom tilknytning til eksterne enheter som en spenningsforsyning.
- **Akustisk (mekanisk) støy og interferens**. Her kan for eksempel uønskede vibrasjoner i selve målesystemet eller i omgivelsene skape problemer ved at de forplanter seg til målesignalet. Dette kan skje ved at enkelte komponenter i målesystemet virker som piezoelektriske mikrofoner.
- **Ukorrekt belastning**. Dette innebærer at målesystemet eller deler av det, for eksempel måleelementet eller de elektroniske komponentene, utsettes for unormale påvirkninger som de ikke er beregnet for. Dette kan være for høy temperatur, for høyt trykk med mere. Hvis disse belastningene er innenfor *driftsgrensene* vil feilen som oppstår være midlertidig, ellers vil også varig skade kunne oppstå. Det er her oftest snakk om systematiske feil.
- **Feil bruk**. Dette innebærer at et system blir brukt feilaktig. Eksempler på dette er for eksempel at måleelementet er feil montert eller brukt, at det er feil i drivspenningen, at det ikke blir tatt hensyn til oppvarmingseffekter ved påslag, eller at det ikke har tilstrekkelig båndbredde slik at dynamiske feil oppstår. Det er også her oftest snakk om systematiske feil.
  Et spesialtilfelle av anvendelsesfeil er at måleelementet forstyrrer prosessen og gjør at inngangsvariabelen får en annen verdi enn den ville ha uten måleelementet til stede.

- **Slitasje- og aldring** (*wear and aging*). Flere målesystemer svekkes med tiden slik at det oppstår drift i eller slitasje av kritiske komponenter.
- **Feil beregning**. Dette er gir oftest systematiske feil og skyldes at verdien på utgangsvariabelen konsekvent blir feil fordi den blir beregnet på feil grunnlag. Det kan for eksempel være et uttrykk hvor det inngår komponentverdier som avviker fra oppgitt verdi. Det kan være at et ikke-lineært system blir antatt å være lineært. Til sist har vi programmeringsfeil som er svært vanlige, men som ofte er for små til at de oppdages selv om de likevel gir utslag.
- **Kvantisering**. Denne feilen er til stede i alle systemer hvor målesignalene kvantiseres (digitaliseres).
- **Avlesning**. Denne feilkilden kan bidra mye til den totale målusikkerheten i instrumenter med analoge visere, men er fraværende ved digital visning av måleresultatet.

Eksempel: Baguette

Det finnes mange eksempler på rare feil som ødelegger for målesystemer. En av de mer spesielle skjedde i 2009 når Large Hadron Collider (LHC) på CERN feilet med en kortslutning på grunn av en baguette. Teorien var at en fugl hadde fløyet over en elektrisk installasjon tilknyttet LHC og sluppet en liten bit baguette ned over installasjonen og dermed forårsaket en kortslutning. Heldigvis hadde det ingen fatale konsekvenser og driften kunne gjenopptas etterpå uten permanente feil.[2]

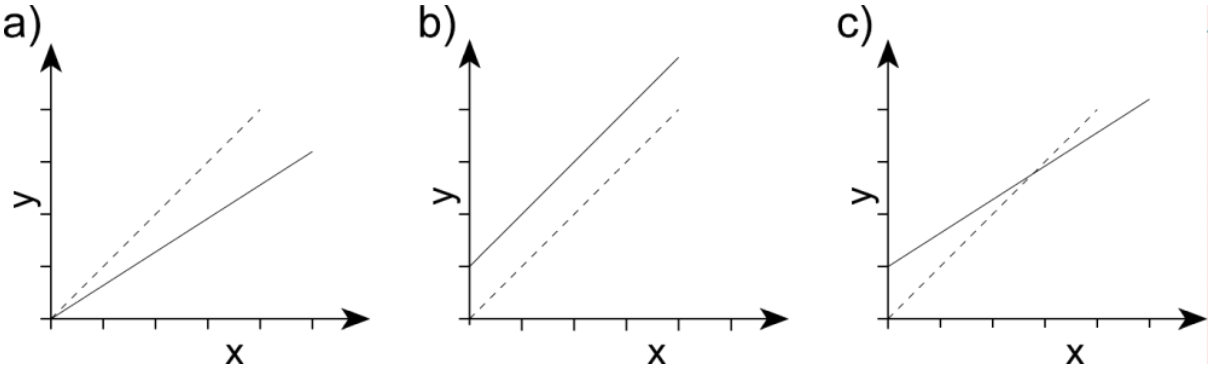
De ulike feilkildene kan virke direkte inn på selve måleprinsippet slik at måleelementet blir lurt. Det er ofte slik at den fremste enden i et målesystem er mest følsom for de ulike typene støy. Men også andre deler av målesystemet, f.eks. elektronikk, kan påvirkes og føre til måleusikkerhet.

Feil som kommer og går på en uforutsigbar måte, kalles for *ambulerende feil*. Denne typen feil kan være svært vanskelig å lokalisere. Feilsøking er forøvrig oftest en omstendelig og komplisert prosess hvor systematikk er viktig for godt resultat. Dette innebærer blant annet å lokalisere feilen ved å systematisk eliminere mulige feilkilder.

3.5 Innjustering og kalibrering

De fleste målesystemer er over tid utsatt for drift eller forandring i statistiske, deterministiske og spesielt systematiske feil. Sistnevnte gjør at karakteristikken, det vil si forholdet mellom systemets utgangs- og inngangsvariabel y/x, forandrer seg. Dette gir seg vanligvis utslag i *forsterkningsfeil* (gain error), *nullfeil* (offset) eller en kombinasjon av disse, se Figur 3. I tillegg kan også systemets linearitetsegenskaper forandre seg.

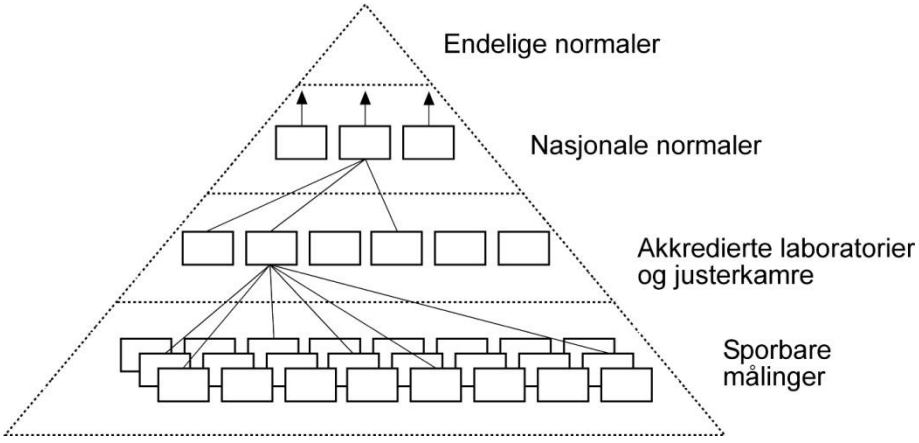
[2] http://blogs.nature.com/news/2009/11/exclusive_interview_baguette_b.html



Figur 3: Eksempler på a) forsterkningsfeil, b) nullfeil og c) begge deler i forhold til den ideelle karakteristikken som er vist med den stiplede linjen.

Forsterknings- og nullfeil kan elimineres ved *kalibrering*. Da bestemmes den statiske karakteristikken til et målesystem, eller deler av et målesystem, gjennom samtidige middelverdimålinger av inngangsvariabelen x og utgangsvariabelen y. Det gjøres samtidig referansemålinger av x og andre omgivelsesvariabler som kan virke inn på y slik at man har kontrollerte forhold og kjente betingelser. Referanseinstrumentene kalles *normaler* (standards). Kalibreringen skjer ved faste prosedyrer som også innebærer at verdiene til inngangsvariabelen endres både oppover og nedover over hele måleområdet.

Ofte er det tilstrekkelig å bruke resultatene fra kalibreringer til å korrigere programvaren for forsterknings- og nullfeil, og la måleomformeren være uforandret. Andre ganger er det ønskelig å korrigere selve måleomformeren slik at forsterknings- og nullfeil justeres bort, noe som i praksis ofte gjøres ved hjelp av to potensiometre. Dette kalles *innjustering*, men blir i dagligtale også ukorrekt omtalt som kalibrering. Etter en innjustering utføres gjerne en ny kalibrering. Sistnevnte er viktig for å ivareta historikken, det vil si å kunne sammenligne nye målinger mot gamle.



Figur 4: Illustrasjon av sporbarhetsstigen og hierarkiet som sikrer sporbare målinger for kalibrerte instrumenter.

Kalibrering er viktig i sammenhenger der måleusikkerheten skal reduseres til et minimum, for eksempel ved fiskale målinger (målinger ved kjøp og salg), ved regulering av kritiske prosesser og ikke minst innen forskning. I slike tilfeller kan *akkrediterte normaler* (referanseinstrumenter) brukes ved regelmessige kalibreringer. Måleusikkerheten til normaler er kjent fra videre kalibreringer oppover mot *nasjonale normaler* som Justervesenet har ansvaret for i Norge. Disse er videre kalibrert mot *endelige normaler* som har lavest måleusikkerhet og oppbevares hos *BIPM* (Bureau

International de Poids et Mesure) i Paris. Når et målesystem er kalibrert etter denne hierarkiske strukturen, er målingene sporbare. Det vil si at nøyaktigheten eller måleusikkerheten kan spores ubrutt mot den endelige standarden i en *sporbarhetsstige* med økende nøyaktighet, se Figur 4.

3.6 Nøyaktighet og repeterbarhet

Som tidligere nevnt er det sjelden at måleusikkerheten blir oppgitt for en sensor, måleomformer eller et målesystem. Derimot kan man finne spesifikasjoner for linearitet, nøyaktighet og repeterbarhet.

- **Nøyaktigheten** til et målesystem angir hvor godt måleverdiene til systemet samsvarer med systemets kalibrerte karakteristikk som representerer sanne verdier av inngangsvariabelen. Nøyaktigheten omfatter derfor alle feil, også systematiske.
- **Repeterbarhet** er definert som det største avviket mellom utgangsverdier fra flere etterfølgende målinger av samme inngangsverdi under identiske forhold og over forholdsvis kort tid. Repeterbarheten omfatter med andre ord statistiske feil og interferens hvis dette er til stede, men ikke systematiske feil.

Siden repeterbarhet ikke omfatter systematiske feil gjelder følgende utsagn:

- Dårlig repeterbarhet medfører dårlig nøyaktighet.
- God nøyaktighet medfører god repeterbarhet.
- God repeterbarhet medfører ikke nødvendigvis god nøyaktighet!

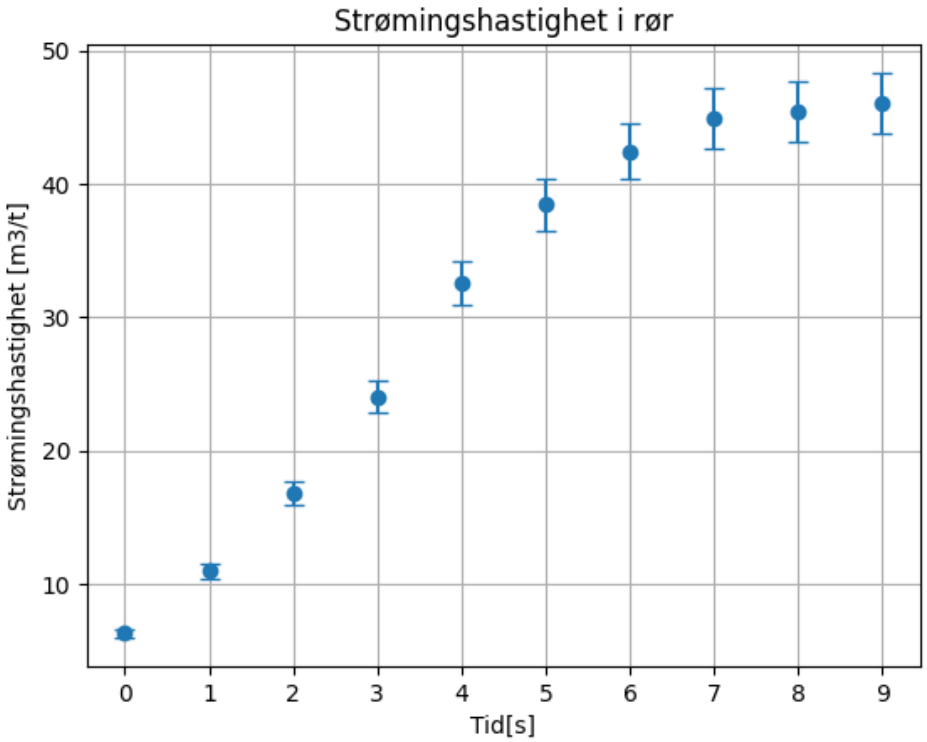
3.7 Presentasjon av måleresultat

Tabell 2 viser i de to første kolonnene rådata slik de foreligger i en datafil. I tredje kolonne er den totale måleusikkerheten presentert ut fra beregninger som ikke er vist her. Måleresultatene presentert i den fjerde kolonne med én desimal sammen med den tilhørende usikkerheten.

Tabell 2: Eksempel på presentasjon av måledata fra en volumstrømningsmåler i tabell.

Tid [s]	Rådata [m^3/t]	Måleusikkerhet [m^3/t]	Presenterte data [m^3/t]
0	6,2783	0,31	6,3 ± 0,3
1	10,9830	0,55	11,0 ± 0,6
2	16,7989	0,83	16,8 ± 0,8
3	24,0094	1,20	24,0 ± 1,2
4	32,5893	1,63	32,6 ± 1,6
5	38,4320	1,92	38,4 ± 1,9
6	42,4432	2,12	42,4 ± 2,1
7	44,8923	2,24	44,9 ± 2,2
8	45,4329	2,27	45,4 ± 2,3
9	46,0392	2,30	46,0 ± 2,3

Måledataene kan også fremstilles grafisk slik det er vist i Figur 5. Koden for plottet finnes i Listing 1. Måleusikkerheten for hvert punkt er angitt med en vertikal stolpe.



Figur 5: Eksempel på grafisk presentasjon av måledata.

```
import numpy as np
import matplotlib.pyplot as plt

tid = np.arange(0, 10);
strm =
np.array([6.2783,10.9830,16.7989,24.0094,32.5893,38.4320,42.4432,44.8923,45.4329,46.0392]
);
std = np.array([0.31,0.55,0.83,1.20,1.63,1.92,2.12,2.24,2.27,2.30]);
plt.errorbar(tid, strm, std, fmt='o', capsize=4);
plt.grid();
plt.xticks(tid);
plt.xlabel("Tid[s]");
plt.ylabel("Strømingshastighet [m3/t]");
plt.title("Strømingshastighet i rør");
```

Listing 1: Python kode for å plotte Figur 5. Dette er skrevet i Jupyter notebook.

4 Digitalisering

Moderne måle- og reguleringssystemer inneholder nå nesten uten unntak en digital enhet som gjør det nødvendig å omforme signaler mellom analog og digital form; signalet må *digitaliseres*. Dette betyr at to ting må gjøres med det kontinuerlige *analoge* målesignalet. Dette er:

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

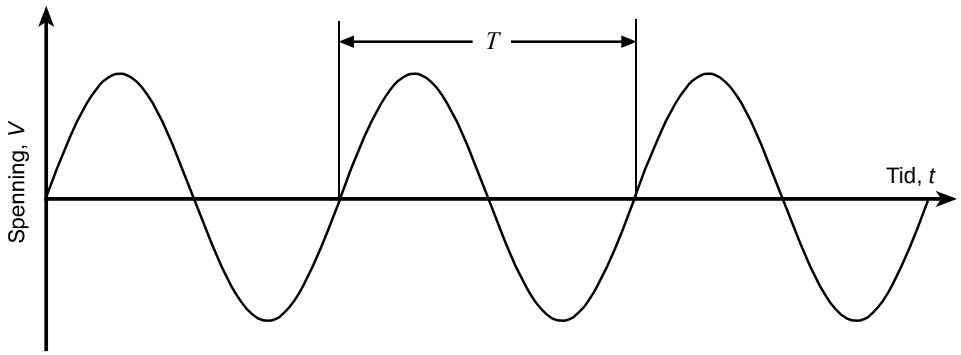
Utfordringen ved digitalisering er å gjøre samplingen så ofte og kvantiseringen med så god oppløsning at informasjon ikke går tapt, men heller ikke oftere eller bedre enn nødvendig; på den måten unngår vi at det skapes unødvendig store datamengder. Vi skal se senere at det også er en grense for hvor tett det er mulig å sample et analogt signal. Oppsummert ønsker vi at det opprinnelige kontinuerlige signalet må kunne rekonstrueres på grunnlag av det digitaliserte signalet. Tiden mellom hver kvantisering kalles sampletiden eller samplingsintervallet, ΔT. Hvis ΔT 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 kontinuerlig signal kan representeres med, og bli rekonstruert fra, et sett med samplede verdier når antall samplinger i sekundet er minst det dobbelte av den høyeste frekvenskomponent til stede i signalet":

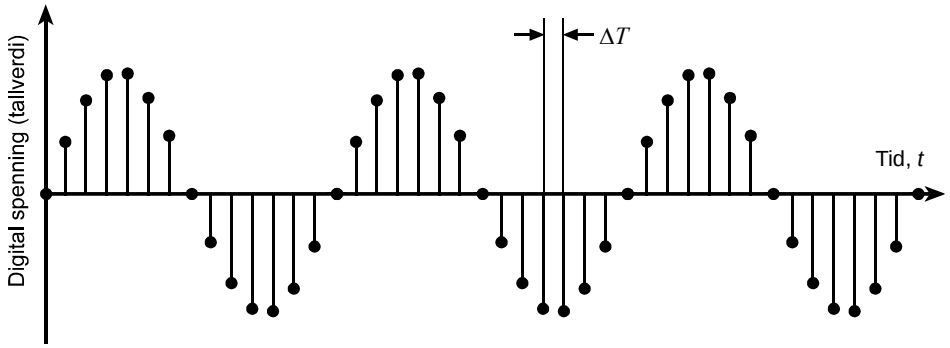
$$f_s = \frac{1}{\Delta T} = 2f_{maks} \tag{2}$$

hvor f_s er samplingsfrekvensen og f_{maks} det analoge signalets høyeste frekvenskomponent. I praksis er f_{maks} den høyeste frekvenskomponent som ikke kan neglisjeres med hensyn på bidrag til signalets totale effekt.

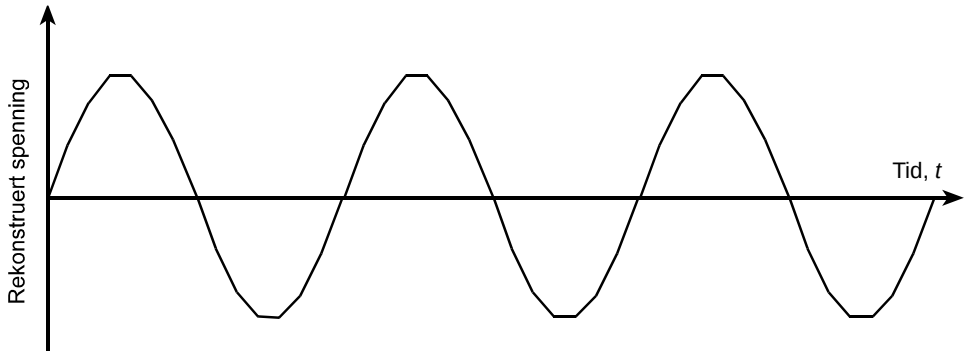
Et kontinuerlig signal eksisterer ved alle tidspunkt og alle spenninger mellom amplitudeverdiene slik det er illustrert med sinussignalet i Figur 6. Dette er også et periodisk signal fordi det gjentas med en periode kalt signalperioden (T). Når vi sampler dette signalet med en fast samplingstid (ΔT) blir resultatet et diskrét signal som vist i Figur 7. Dette eksisterer kun ved faste tidspunkt. Men hvis Nyquists samplingsteorem er oppfylt så kan det diskrete signalet rekonstrueres ved at man antar hvilke verdier signalet har mellom hver sampling. Da vil også den observerte frekvensen vi leser av det rekonstruerte signalet stemme overens med signalfrekvensen. En av de enkleste formene for rekonstruksjon er å trekke rette linjer mellom fra hvert samplingspunkt til det neste slik det er vist i Figur 8. Det finnes mer avanserte former for rekonstruksjon, for eksempel ved å bruke kurvetilpasning i stedet for en rett linje mellom hver sampling. I alle tilfeller antar vi da noe om det opprinnelige signalet når vi rekonstruerer det.



Figur 6: Et kontinuerlig, periodisk (analogt) signal som eksisterer for alle verdier av tiden t. Signalfrekvensen er f = 1/T*, der* T *er periodetiden*



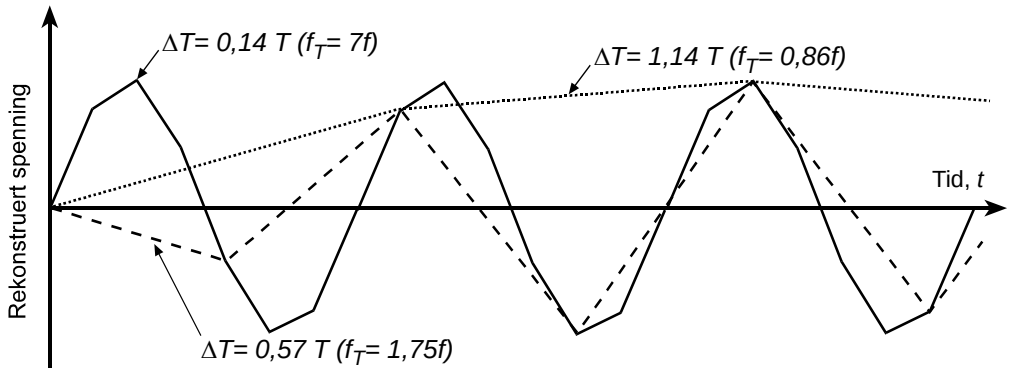
Figur 7: Ved å sample signalet i med en analog-til-digital-omformer får vi et diskré (digitalt) signal som kun eksisterer i bestemte tidspunkt separert med sampletiden ΔT.



Figur 8: En svært enkel rekonstruksjon (rette linjer mellom samplepunktene) av det samplede signalet i Figur 7. Her er ΔT = 0,07T

Hvis samplingstiden blir for lang i forhold til signalperioden, altså at Nyquists samplingsteorem ikke er oppfylt, så vil det opprinnelige signalet ikke kunne rekonstrueres samtidig som den observerte signalfrekvensen heller ikke stemmer overens med det opprinnelige signalets frekvens. Dette er vist for forskjellige sampletider og rettlinje-rekonstruksjon i Figur 9, spesielt rekonstruksjonen med den lengste sampletiden. Ut fra denne rekonstruksjonen ser det ut til at signalets frekvens er langt lavere enn den opprinnelige. Hvis for eksempel et periodisk signal samples nøyaktig én gang per periode, så vil det rekonstruerte signalet bli en horisontal linje, altså frekvens lik 0. Dette kalles *nedfolding* eller *aliasing* og oppstår når Nyquists samplingsteorem ikke er oppfylt. Det rekonstruerte signalet ligner ikke det opprinnelige og har heller ikke samme frekvens. Begrepet nedfolding brukes siden signalfrekvensen er "*foldet*" rundt halve

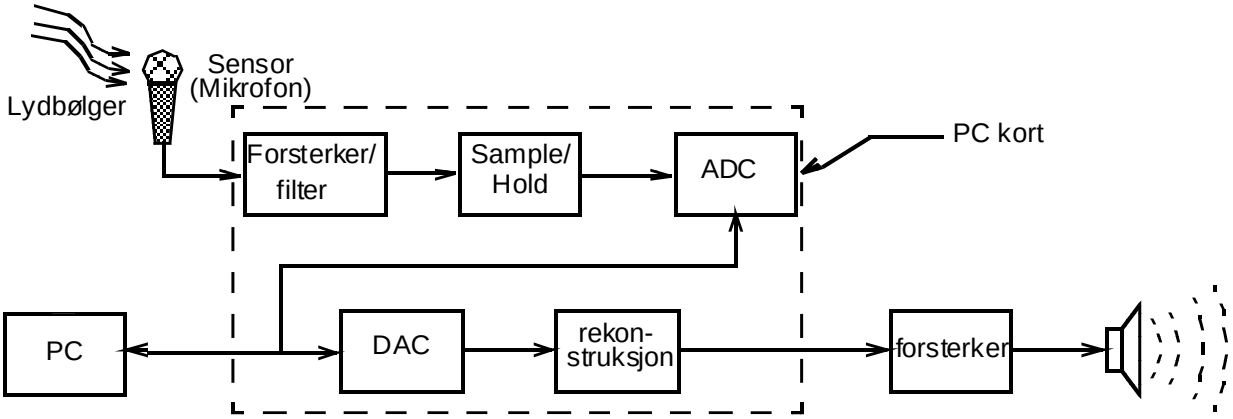
samplefrekvensen, det vil si at signalets frekvens tilsynelatende ligger like langt under halve samplefrekvensen ($f_s/2$) som den i virkeligheten ligger over.



*Figur 9 Eksempel på rett-linje-rekonstruksjon med utgangspunkt i det kontinuerlige signalet over, men med lenger sampletider enn i Figur 7.**Error! Reference source not found.***

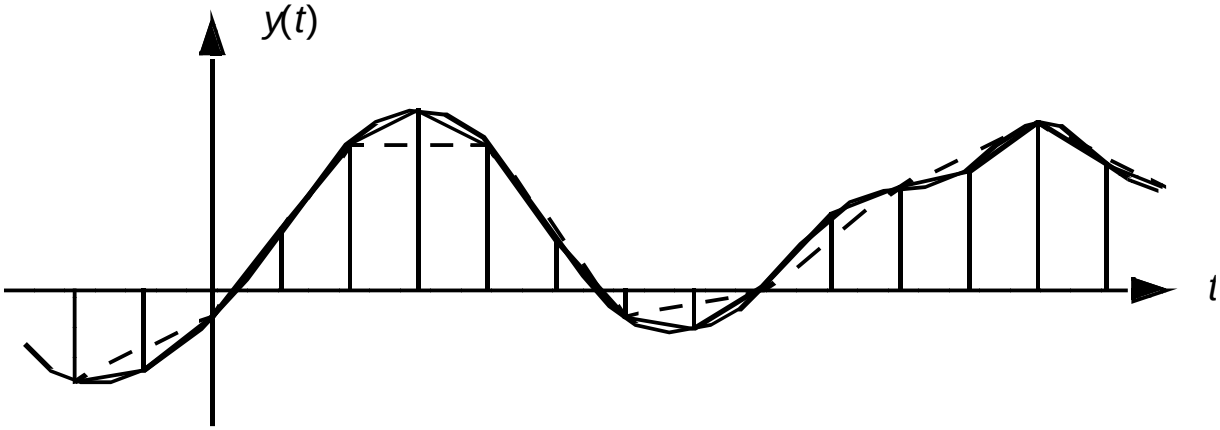
4.1 Eksempel på Nyquists Samplingsteorem

Nyquists samplingsteorem kan illustreres med et eksempel som demonstrerer hvorfor digital lyd er mulig. Denne teknologien benytter seg jo av data som er samplet. Vi kan bruke en PC med nødvendig ekstrautstyr både for å samle inn data; digitalisere lyden, og for å rekonstruere de digitale dataene til et analogt inngangssignal til en høyttaler, se eksempelet i Figur 10.



Figur 10: Skjematisk fremstilling av et PC-basert måle- og rekonstruksjonssystem for audiosignaler.

For at den menneskelige hjerne skal kunne oppfatte det digitaliserte lydsignalet må de samplede verdiene settes sammen i en kontinuerlig kurve. Dette gjøres altså ved rekonstruksjon hvor vi interpolerer mellom samplepunktene. Den enkleste metoden er rett og slett å holde den samplede verdien konstant inntil neste sampling. De samplede verdiene omdannes da til en trappetrinnkurve. Rekonstruksjon med rett linje er bedre, og dette er illustrert i Figur 11. Her er den heltrukne kurven signalet slik det så ut før det ble samplet, endepunktene på de vertikale linjene er de samplede verdiene, den heltrukne rette linjen mellom samplepunktene er signalet rekonstruert ved lineær interpolering og den stiplete linjen representerer en tilsvarende rekonstruksjon der bare hver annen sampling er brukt. Det er tydelig fra Figur 11 at den heltrukne rette linjen er forskjellig fra det opprinnelige signalet slik at vi har fått en viss forvrengning i det rekonstruerte signalet. Brukes kun annenhver sampling i rekonstruksjonen er det klart at forvrengningen er større (den stiplete linjen). Det virker som om jo tettere samplepunktene kan legges jo mindre blir forvrengningen.

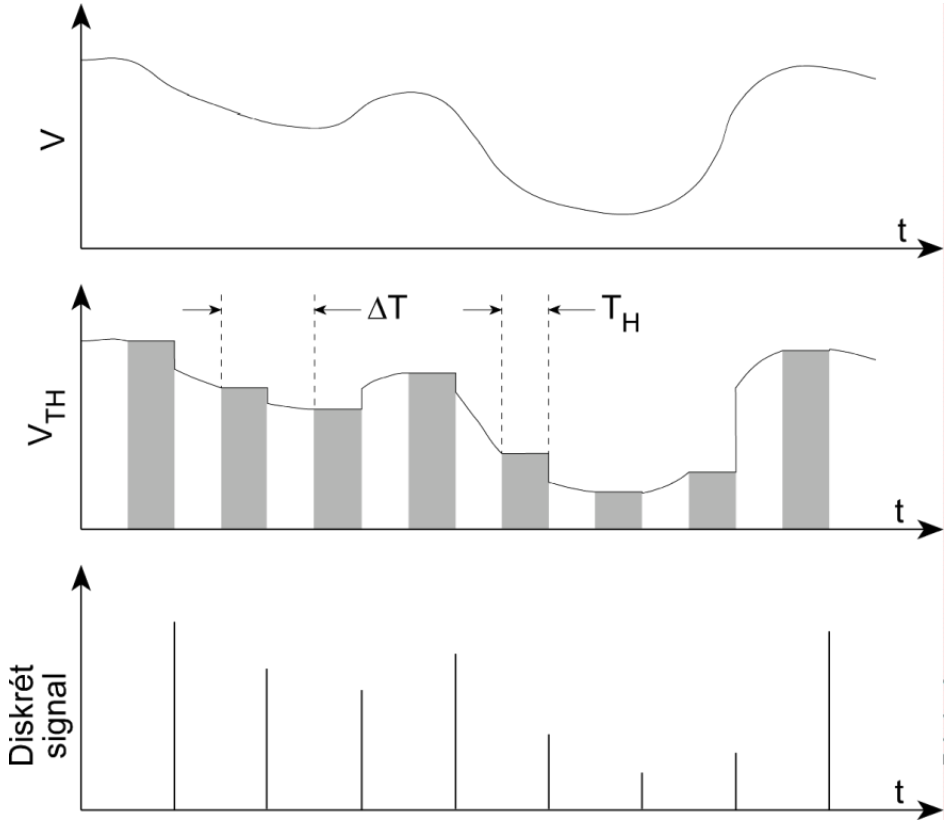


Figur 11: Utsnitt av digitalisert audiosignal

Et menneske oppfatter ikke lyd med frekvens mye over 20 kHz. Vi kan med et elektronisk filter begrense signalets frekvensomfang (vi kaller dette normalt signalets båndbredde) til 20 kHz og sample med ca. 40 kHz. Samplingsfrekvensen på audio-CD-plater er 44,1 kHz mens antall verdier signalet representeres med er 65536 (2^{16}). Og alle som har hørt på CD-lyd vet at dette er tilstrekkelig og at dette derfor fungerer i praksis. Samme resonnement gjelder også for andre anvendelser som digital video.

4.2 Analog-til-digital omforming

Selve kvantiseringen, det å omforme en analog spenning til en digital verdi, kalles analog-til -digital omforming eller analog-til-digital konvertering.



Figur 12: Plottet i midten illustrerer utgangssignalet fra en SH-krets med et inngangssignal som vist øverst. Tiden SH-kretsen holder signalet konstant er vist med gråtone. Verdiene i det diskrete signalet foreligger, som antydet, med en tidsforsinkelse T_H etter selve samplingen.

Denne prosessen kan ta opptil flere millisekunder avhengig av hvilken type elektronikk som benyttes. Av den grunn er det ofte nødvendig å holde den analoge spenningen som skal digitaliseres, konstant så lenge omformingen pågår. Til dette benyttes en SH-krets (*Sample og Hold*) som låser inngangsspenningen og holder den konstant så lang tid, T_H, som konverteringen tar. Dette er illustrert i Figur 12. SH-kretsen som videre er knyttet til en ADC (*Analogue to Digital Converter*) hvor selve konverteringen gjøres, styres som regel med et eksternt klokkesignal slik at samplingen gjøres ved faste tidsintervall, ΔT. Tiden signalet holdes konstant, T_H, må tilpasses tidsforbruket til ADCen, slik at den blir kortest mulig. Ofte blir SH-kretsen automatisk slått av av ADCen med en gang konverteringen er unnagjort. Det betyr, som vist i Figur 12, at de digitale verdiene av et analogt signal foreligger etter en tidsforsinkelse lik T_H.

Det finnes tre kategorier ADCer:

- "*Flash*"*-omformeren* er den desidert raskeste ADCen med en konverteringstid på typisk 20 ns, men den har de dårligste linearitetsegenskapene.
- En *suksessiv-approksimasjon-omformer* har bedre linearitetsegenskaper og en konverteringstid på rundt 20 μs.
- Til sist følger *dobbel-rampe-omformeren* som bruker typisk 20 ms på en konvertering, men er til gjengjeld den ADCen som har best støyimmunitet og de beste linearitets-egenskaper.

De vanligste datainnsamlingskortene bruker suksessiv-approksimasjon-omformere med en 12-bits oppløsning på 4096 (2^{12}). Det vil si at det analoge inngangssignalets verdi blir representert med et tall mellom 0 og 4095.

5 Datainnsamlingsenhet NI myDAQ

National Instruments myDAQ er en bærbar (portabel) datainnsamlingsenhet (DAQ) basert på standard USB-grensesnitt som bruker National Instruments LabVIEW for datainnsamling og analyse av eksperimentelle måleoppsett. Figur 13 viser et bilde av en NI myDAQ-enhet.

For mer utdypende informasjon om myDAQ enn gitt her, henvises det til brukermanualene fra National Instruments [2] [3]

Figur 13: NI myDAQ datainnsamlingsenhet

5.1 Spesifikasjon

Datainnsamlingsenheten NI myDAQ har følgende spesifikasjoner:

Analog input

- Antall kanaler: 2 differensielle eller 1 stereo audio inngang
- ADC oppløsning: 16 bit
- Maksimum samplings (punktprøvings) rate: 200 kS/s
- Timing nøyaktighet: 100 ppm of sample rate
- Timing oppløsning: 10 ns
- Måleområde (range) analog inngang: ±10 V, ± 2V, DC-coupled
- Måleområde (range) audio inngang: ± 2V, AC-coupled
- Tilkobling analog inngang: skru-terminaler
- Tilkobling audio inngang: 3.5 mm stereo jack
- Inngangsimpedans: > 10 GΩ || 100 pF

Analog output

- Antall kanaler: 2 med referanse til jord-potensial eller en stereo audio utgang
- DAC oppløsning: 16 bit
- Maksimum oppdateringsrate: 200 kS/s
- Måleområde analog utgang: ±10 V, ± 2V, DC-coupled
- Måleområde audio utgang: ± 2V, AC-coupled
- Maksimum utgangsstrøm (analog utgang): 2 mA
- Utgangsimpedans analog utgang: 1 Ω
- Utgangsimpedans audio utgang: 120 Ω
- Tilkobling analog utgang: skru-terminaler
- Tilkobling audio utgang: 3.5 mm stereo jack
- Timing nøyaktighet: 100 ppm of sample rate
- Timing oppløsning: 10 ns

Digital input/output (I/O)

- Antall digitale signallinjer: 8; DIO <0..7>
- Logic level: 5 V kompatible LVTTL inngang, 3.3 V LVTTL utgang

Teller/ timer

- Antall tellere: 1
- Oppløsning: 32 bit
- Frekvens internklokke: 100 MHz
- Nøyaktighet internklokke: 100 ppm
- Maksimum oppdateringsrate teller og pulsgenerering: 1 MS/s

Digitalt multimeter

- Funksjoner: DC spenning, AC spenning, DC strøm, AC strøm, resistans, diode
- Oppløsning 3.5 digits
- Inngangskobling: DC (DC spenning, DC strøm, resistans, diode); AC (AC spenning, AC strøm)
- Spesifikasjon for spenningsmåling er gitt i Tabell 3
- Spesifikasjon for strømmåling er gitt i Tabell 4
- Spesifikasjon for motstandsmåling er gitt i Tabell 5

Tabell 3: Spesifikasjon for spenningsmåling med myDAQ multimeter [3]

Voltage Measurement

DC ranges..... 200 mV, 2 V, 20 V, 60 V

AC ranges..... 200 mV_{rms}, 2 V_{rms}, 20 V_{rms}

Note All AC voltage accuracy specifications apply to signal amplitudes greater than 5% of range.

Accuracy

Function	Range	Resolution	Accuracy ± ([% of Reading] + Offset)	
DC Volts	200.0 mV	0.1 mV	0.5% + 0.2 mV	
	2.000 V	0.001 V	0.5% + 2 mV	
	20.00 V	0.01 V	0.5% + 20 mV	
	60.0 V	0.1 V	0.5% + 200 mV	
			40 to 400 Hz	**400 to 2,000 Hz**
AC Volts	200.0 mV	0.1 mV	1.4% + 0.6 mV*	—
	2.000 V	0.001 V	1.4% + 0.005 V	5.4% + 0.005 V
	20.00 V	0.01 V	1.5% + 0.05 V	5.5% + 0.05 V

* The accuracy for AC Volts 200.0 mV range is in the frequency range of 40 Hz to 100 Hz. For example, for a 10 V using the DC Volts function in the 20.00 V range, calculate the accuracy using the following equation:

10 V × 0.5% + 20 mV = 0.07 V

Input impedance..... 10 MΩ

Tabell 4: Spesifikasjon for strømmåling med myDAQ multimeter [3]

Current Measurement

DC ranges .....20 mA, 200 mA, 1 A

AC ranges .....20 mA_{rms}, 200 mA_{rms}, 1 A_{rms}

Note All AC accuracy specifications within 20 mA and 200 mA ranges apply to signal amplitudes greater than 5% of range. All AC accuracy specifications within the 1 A range apply to signal amplitudes greater than 10% of range.

Accuracy

Function	Range	Resolution	Accuracy ± ([% of Reading] + Offset)	
DC Amps	20.00 mA	0.01 mA	0.5% + 0.03 mA	
	200.0 mA	0.1 mA	0.5% + 0.3 mA	
	1.000 A	0.001 A	0.5% + 3 mA	
			40 to 400 Hz	**400 to 2,000 Hz**
AC Amps	20.00 mA	0.01 mA	1.4% + 0.06 mA	5% + 0.06 mA
	200.0 mA	0.1 mA	1.5% + 0.8 mA	5% + 0.8 mA
	1.000 A	0.001 A	1.6% + 6 mA	5% + 6 mA

Input protection.....Internal ceramic fuse, 1.25 A 250 V, fast-acting, 5 × 20 mm, F 1.25A H 250V (Littelfuse part number 02161.25)

Tabell 5: Spesifikasjon for motstandsmåling med myDAQ multimeter [3]

Resistance Measurement

Ranges ..... 200 Ω, 2 kΩ, 20 kΩ, 200 kΩ, 2 MΩ, 20 MΩ

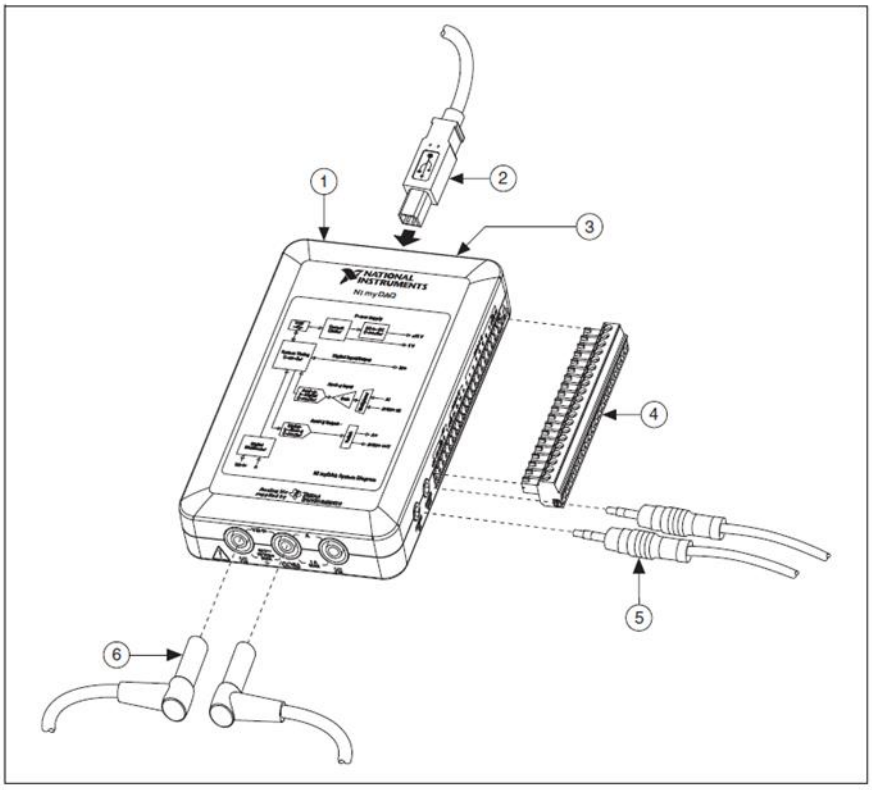
Accuracy

Function	Range	Resolution	Accuracy ± ([% of Reading] + Offset)
Ω	200.0 Ω	0.1 Ω	0.8% + 0.3 Ω*
	2.000 kΩ	0.001 kΩ	0.8% + 3 Ω
	20.00 kΩ	0.01 kΩ	0.8% + 30 Ω
	200.0 kΩ	0.1 kΩ	0.8% + 300 Ω
	2.000 MΩ	0.001 MΩ	0.8% + 3 kΩ
	20.00 MΩ	0.01 MΩ	1.5% + 50 kΩ

* Exclusive of lead wire resistance

5.2 Oppkobling av NI myDAQ

Mulig oppkobling av NI myDAQ er vist i Figur 14.



1	NI myDAQ	4	20-Position Screw Terminal Connector
2	USB Cable	5	Audio Cable
3	LED	6	DMM Banana Cable

Figur 14: Skisse over hvordan oppkobling mot NI myDAQ skal gjøres. Hentet fra [2]

5.3 Signaltilkobling

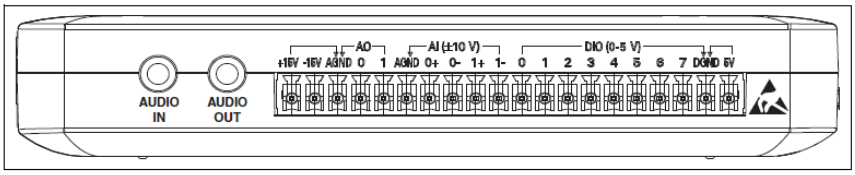


Figure 4. NI myDAQ 20-Position Screw Terminal I/O Connector

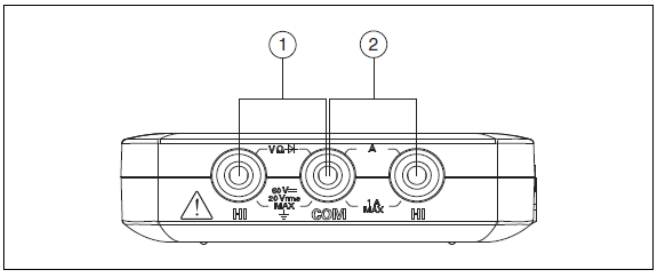
Table 1. Screw Terminal Signal Descriptions

Signal Name	Reference	Direction	Description
AUDIO IN	—	Input	**Audio Input**—Left and right audio inputs on a stereo connector
AUDIO OUT	—	Output	**Audio Output**—Left and right audio outputs on a stereo connector
+15V/–15V	AGND	Output	+15 V/–15 V power supplies
AGND	—	—	**Analog Ground**—Reference terminal for AI, AO, +15 V, and –15 V
AO 0/AO 1	AGND	Output	Analog Output Channels 0 and 1
AI 0+/AI 0–; AI 1+/AI 1–	AGND	Input	Analog Input Channels 0 and 1
DIO <0..7>	DGND	Input or Output	**Digital I/O Signals**—General-purpose digital lines or counter signals
DGND	—	—	**Digital Ground**—Reference for the DIO lines and the +5 V supply
5V	DGND	Output	5 V power supply

Figur 15: Definisjon av tilkoblingspunkter på rekkekontakten. Fra [2]

Figur 15 viser de forskjellige tilkoblingspunktene på rekkekontakten. I denne laben skal vi primært benytte *Ai0+* og *Ai0-*, samt at vi vil bruke myDAQen som spenningskilde, og dermed ta i bruk jordkontakten og *5V* utgangen.

5.4 Tilkobling for multimetermålinger



1 Connectors for Voltage/Resistance/Diode/Continuity
2 Connectors for Current

Figure 5. Connections for DMM Measurements

Table 2. DMM Signal Descriptions

Signal Name	Reference	Direction	Description
HI (VΩ→▶⊢)	COM	Input	Positive terminal for voltage, resistance, and diode measurements
COM	—	—	Reference for all DMM measurements
HI (A)	COM	Input	Positive terminal for current measurements (Fused: F 1.25 A 250 V Fast-Acting)

Figur 16: Tilkobling for bruk av digitalmultimeteret. Fra [2]

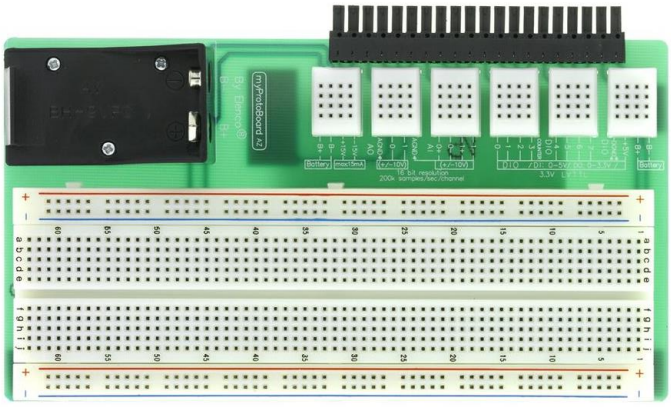
Figur 16 viser tilkoblingspunktene for multimeteret. Multimeteret kan brukes sammen med en multimeterapplikasjon på datamaskinen.

5.5 Tilkoblingsenheter for NI myDAQ

To tilkoblingsenheter for NI myDAQ er tilgjengelig på PHYS114-laboratoriet.

1.1.1 myProtoBoard

myProtoBoard fra Elenco (USA) er et koblingsbrett for elektriske komponenter som kan kobles til NI myDAQ via rekkekontakten, som vist i Figur 17.



Figur 17: myProtoBoard tilkoblingsenhet for NI myDAQ

myProtoBoard er utstyrt med tilkobling (female headers) for NI myDAQ, Tilkoblingskortet har størrelse 16,5 x 5,4 cm, og et 9 V batteri kan installeres på enheten for ekstra tilgang på strøm og spenning i tillegg til standard tilgang på strøm og spenning fra NI myDAQ.

1.1.2 BNC adaptor board

BNC adaptor board (FRI-2000-2) fra Florida Research Instruments (USA) er en tilkoblingsenhet for NI myDAQ som gjør 2 analoge innganger og 2 analoge utganger tilgjengelige for BNC tilkobling, som vist i Figur 18.

Figur 18: BNC adaptor board (FRI-2000-2) tilkoblingsenhet for NI myDAQ

I tillegg til standard BNC- tilkobling gir *BNC adaptor board* også tilgang til NI myDAQs digitale kanaler samt et utvalg testpunkter.

6 Datainnsamling ved hjelp av Python

I denne laboppgaven skal vi bruke et API (Application Programming Interface) fra National Instruments som gjør at vi kan bruke Python til å gjøre datainnsamling fra NI myDAQ.

Python er et programmeringsspråk som ble laget av Guido van Rossum og sluppet i første utgave i 1991. En viktig filosofi bak Python er å øke lesbarheten til kode, og en av måtene dette er løst på er for eksempel at mellomrom (*blank spaces*) og innrykk har betydning for tolkningen av språket.

Python er beregnet på utvikling av verktøy og applikasjoner. Det er et imperativt, objektorientert og fortolket høynivåspråk. Typiske bruksområder er automatisering, dataintegrering og dataanalyse.

En av de store fordelene med Python er at det har et rikt standardbibliotek, samt at det finnes svært mange biblioteker som man kan utvide språket med. Dette gjør at man har veldig mange funksjoner tilgjengelig om man for eksempel vil gjøre databehandling, matematisk og statistisk beregning, og plotting og presentasjon av data. Utfordringen knyttet til dette er selvsagt det å kjenne til alle mulighetene språket har slik at man velger den løsningen som er best for oppgaven man skal løse.

Dette kapitlet vil inneholder en kort introduksjon til python, til utviklingsverktøyet anaconda/spyder og til de to APIene som skal benyttes i laboratorieoppgaven: *pyvisa* og *NI-DAQmx*.. For mer utfyllende informasjon om python generelt anbefales for eksempel den offisielle python dokumentasjonen på nett [4]

6.1 Utviklingsverktøy for Python

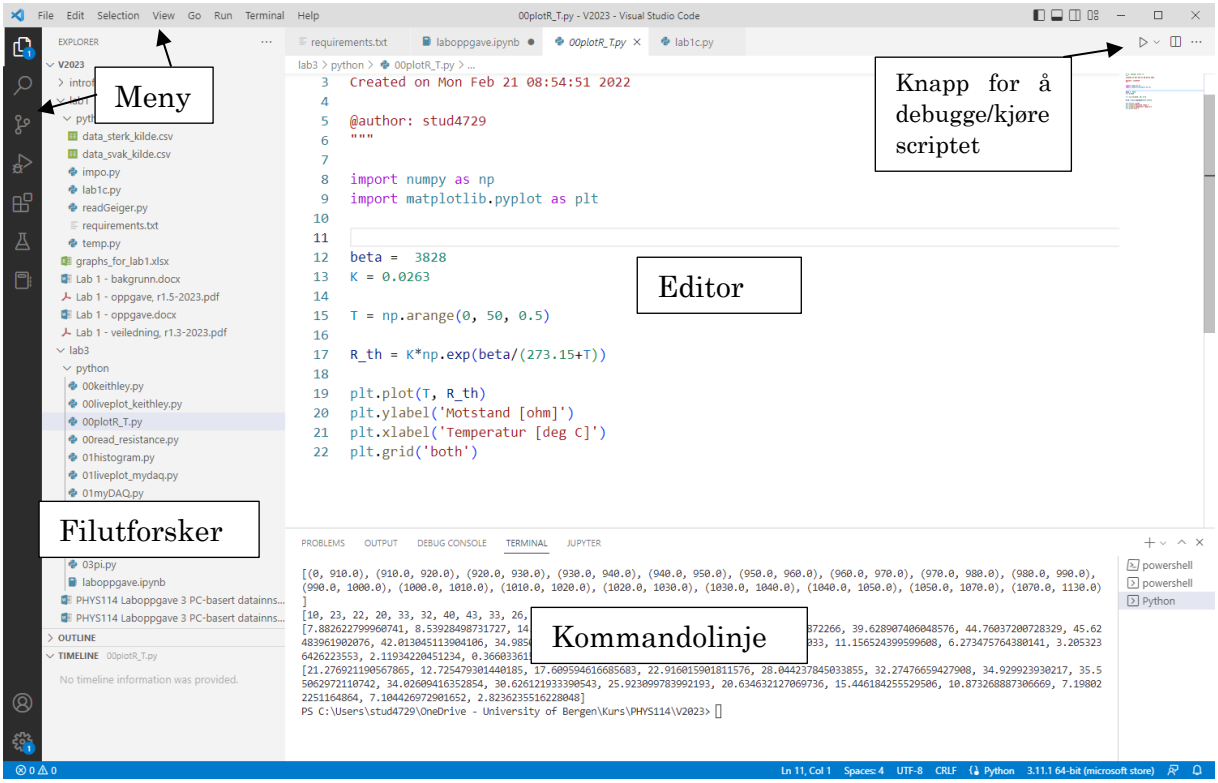
Det finnes mange forskjellige utviklingsverktøy for Python, med sine styrker og svakheter. Disse verktøyene kalles normalt sett for et *Integrated Development Environment (IDE)*. Et IDE skiller seg fra en kode-editor på flere måter. Der en kode-editor er enkel og har et fokus kun på å skrive programmeringskode, har et IDE flere funksjoner som for eksempel verktøy for feilsøking, vanligvis kalt for *debugging*, bygging og distribuering av kode. Vanligvis så vil et IDE inneholde følgende funksjoner:

- En editor for programmeringskode
- En kompilator eller interpreter[3].
- En integrert debugger
- Et grafisk brukergrensesnitt
- Vanligvis også *code completion*. Dette betyr at du kan skrive starten på en konstruksjon, og så fyller den automatisk inn for deg, eller at du får hint om hvilke parametre som gjelder for en funksjon.

Det finnes flere IDE som er gode til å programmere Python, og de fleste er gratis. Eksempler er *IDLE, PyCharm, Visual Studio Code, Spyder, Atom Jupyter, PyDev* og mange flere.

[3] Man trenger en kompilator hvis man skriver pråk som må kompileres til prosessorinstruksjoner for å kunne bli kjørt, f.eks. Java, C++ osv. Skriptspråk, som Python, trenger kun en interpreter som tolker koden.

I PHYS114 anbefaler vi at dere bruke Visual Studio Code (ofte forkortet til VSCode) eller Spyder. Begge verktøyene er installert på labPCene. Både Python og utviklingsverktøyene kommer svært ofte med oppdateringer, så det kan være at eksemplene av VSCode og Spyder i henholdsvis Figur 19 og Figur 20 ser litt annerledes ut hvis dere laster det ned til egen PC.



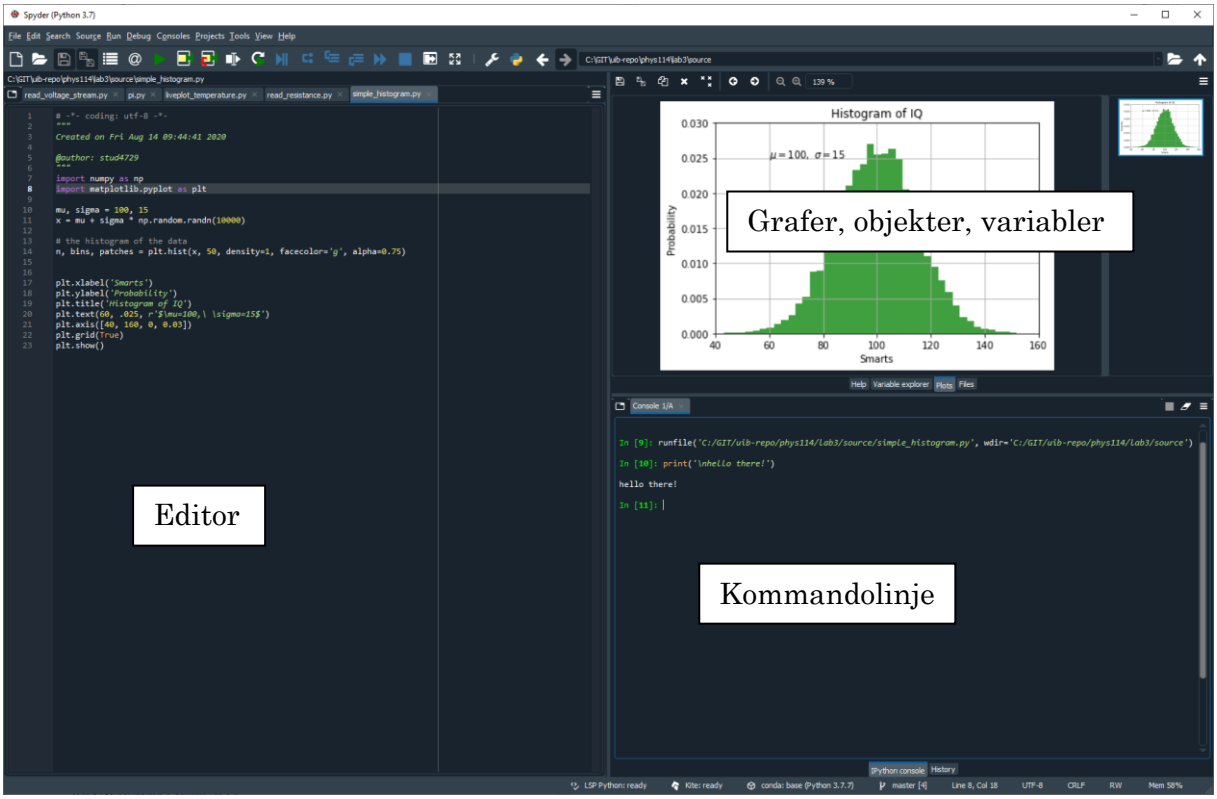
Figur 19: Visual Studio Code utviklingsverktøy.

Den store styrken til VSCode er at man har en svært stor grad av frihet hvordan man kan sette det opp. Det finnes tilleggspakker for mange forskjellige programmeringsspråk, og det anbefales på det sterkeste at man setter opp dette for Python.

I Figur 19 vises en måte å sette opp VSCode på som er hensiktsmessig når man skal programmere i Python. Til venstre er filutforskeren der vi kan selv velge hvilken katalog som skal vises. Hoveddelen av vinduet er editoren. I eksempelet vises bare en kodefil, men man kan ha flere synlige ved siden av hverandre om man ønsker det. I bunnen vises terminalvindu, der vi kan ha flere typer terminaler åpne; en av de er kommandolinje. Her kan man starte programmene man har skrevet, eller man kan f.eks. installere ekstrapakker ved kommandoen: «*python -m pip install -r requirements.txt*». Her er filen *requirements.txt* en tekstfil som inneholder en liste av pakker, f.eks. NumPy, SciPy og så videre.

Når man skal teste kode så er det lettest å bruke knappen oppe til høyre som anvist i figuren. Der kan man velge å kjøre programmet ved å trykke på knappen. Hvis det er feil, så vil feilmeldingene komme opp i terminalvinduet. Hvis man ønsker å i større grad forstå feilen, så kan man velge å debugge koden på den samme knappen.

Det finnes mange tutorials på nettet for å lære seg å bruke VScode sammen med Python. Det anbefales å se litt på disse[4].



Figur 20: Grafisk brukergrensesnitt for Anaconda Spyder version 4.1.4

Anaconda er en gratis *open-source* datavitenskapelig plattform. Den består av Python- og R-distribusjoner, samt en pakkemanager som heter conda. Python og R er forskjellige programmeringsspråk. Anaconda tilbyr svært mange forhåndsinstallerte biblioteker og pakker. Eksempler her er NumPy for matematisk beregning, SciPy for statistisk beregning og Pandas for databehandling.

En av applikasjonen som medfølger anaconda er utviklingsverktøyet Spyder. Spyder gir deg tilgang til en editor, en python kommandolinje, samt en mulighet til å se plott, variable, objekter og verdien til disse. Se Figur 20. Øverst oppe er verktøylinjen, hvor det er knapper for å kjøre script og også debugge de.

Kommandolinjen kan brukes til å skrive kode direkte inn også. For eksempel kan man teste og bruke Python som kalkulator ved å skrive '*2 + 2*' på kommandolinjen og se hva slags verdi man får tilbake. Et annet enkelt eksempel å teste er Hello world eksempel: '*print('Hello world')*'. Begge disse linjene kan også skrives inn i et skript om man ønsker dette (selv om '*2 + 2*' der gir mindre mening uten å tilordne resultatet til en variabel).

I tillegg finnes det *Anaconda Prompt* (eventuelt *Anaconda Powershell Prompt*[5]) som en del av installasjonen. Dette er kommandolinje som man også kan kjøre scripts fra. Et script blir da typisk startet med kommandoen '*python <navn_på_script>.py*".

[4] Et eksempel er på microsofts egne sider: Get Started Tutorial for Python in Visual Studio Code
[5] Forskjellen mellom disse er at powershell prompt emulerer et Linux miljø og dermed kjenner til standard Linux kommandoer

6.2 Programmering med Python

Python kommer i mange forskjellige versjoner, og mellom hoved versjonene, f.eks. Python 2.x versus Python 3.x, kan det være så store forskjeller at versjonene ikke er kompatible med hverandre. Denne seksjonen vil gi noen enkle eksempler på Python 3.x kode da det er denne versjonen som per dags dato er installert på maskinene på laboratoriet.

Kodeeksemplene som er gitt her er alle i form av et skript, som man altså skriver i det venstre vinduet (*editor*) i Spyder (Figur 20). Eksemplene er tatt fra [5], hvis ikke annet er nevnt.

Det er ikke anbefalt at man skriver Python kode direkte i kommandolinjen, foruten hvis man ønsker å teste enkle kodestrukturer. Hvis man for eksempel er usikker på hvordan man kaller en funksjon eller liknende, så kan det være effektivt å prøve den ut på kommandolinjen.

6.2.1 Matematikk

```
x = 1
y = -1
print("(x,y) = (%g,%g)"%(x,y))
```

Listing 2: Kodeeksempel som viser print med printf-formattering

Koden i Listing 2 vil skrive ut *(1, -1).* prosenttegnet etterfulgt av bokstaven *g* bestemmer hvordan verdien av variabelen skal skrives til skjerm. 'g' angir flyttall som blir angitt på eksponentiell form hvis eksponenten er mindre enn -4. Det finnes også mange andre måter å formattere et tall på i python som er gitt i [4].

```
a = -2
b = 3

# Addisjon:
print("a + b =", a+b)

# Subtraksjon
print("a - b =", a-b)

# Multiplikasjon
print("a * b =", a*b)

# Divisjon
print("a / b =", a/b)

# Opphøying
print("a opphøyd i b = a**b = ",a**b)
```

Listing 3: Aritmetiske operasjoner i Python

Listing 3 viser de vanlige matematiske operasjonene man kan utføre i Python. Dette er ganske likt som i tilsvarende språk.

For mer avansert matematikk så bør man bruke biblioteket *numpy* [6]. Dette er et bibliotek som inkluderer definisjoner av viktige konstanter som for eksempel π, samt datastrukturer og funksjoner som er svært effektive til matematiske operasjoner på større datamengder.

6.2.2 Lister

En liste i Python er en samling av elementer. I motsetning til andre språk så kan en liste i Python bestå av elementer med ulik type. Dette betyr at en liste i Python kan inneholde

strings, *ints* og *floats*, eller pekere til funksjoner for eksempel. Listing 4 viser hvordan man definerer en tom liste, for deretter å legge til verdiene 10 og 12 til listen.

```
liste1 = []
liste1.append(10) # Nå har liste kun ett element - nemlig 10
liste1.append(12) # Elementet 12 er nå plassert bak 10
print(liste1)
```

Listing 4: Definisjon av en tom liste, hvor to elementer blir lagt til

Hvis man allerede har noen elementer, så kan disse legges inn i en liste som vist i Listing 5.

```
a = 1
b = sum # nå er b en funksjon - nemlig den innebygde funksjonen sum.
c = "hei"

liste2 = [a,b,c]
```

Listing 5: Liste laget av eksisterende elementer av forskjellig type.

Hvis man skal hente ut elementer fra en liste så ligger første element på 0'te plass. Det vil si at koden i Listing 6 vil skrive ut *10* og *hei*.

```
print(liste1[0]) # skriver ut 10
print(liste2[2]) # skriver ut "hei"
```

Listing 6: Utskrift av enkel-elementer i liste1 og i liste2.

For matematiske operasjoner kan det med stor fordel benyttes et *numpy array*. Et *numpy array* er homogent, det vil si at kun elementer av samme type kan lagres. I tillegg finnes det svært mange matematiske metoder som fungerer svært effektivt på et *numpy array*. For eksempel så elementene veldig enkelt sorteres med innbygde sorteringsalgoritmer, man kan gjøre operasjoner på alle elementene i *arrayet* uten å bruke løkker og så videre.

Et enkelt eksempel på å lage et numpy array, og hvordan dette ser ut er gitt i henholdsvis Listing 7 og Figur 21. Eksempelet er hentet fra [6]. Første linjen er nødvendig for å importere *numpy* biblioteket inn i vårt kodemiljø.

```
import numpy as np
a = np.array([1, 2, 3])
```

Listing 7: Bruk av numpy for å lage et 1D array

Figur 21: Arrayet i definert Listing 7.

6.2.3 Løkker

Løkker er et nyttig verktøy i programmering og det er to hovedtyper: *for-løkker* og *while-løkker*. *For-løkker* brukes typisk når man løper over en struktur med kjent lengde, mens en *while-løkke* vil kjøre så lenge en boolsk test evalueres til *sann*.

```
for i in range(10,22,2):
    print(i)
```
Listing 8: eksempel på for løkke

Listing 8 gir et enkelt eksempel på en for løkke. Denne løkken benytter *range* funksjonen, for å angi et gyldighetsområde hvor løkken skal jobbe over. I dette tilfellet er det fra 10 til 22. 2 tallet angir størrelsen på steget man tar. Dette betyr at denne koden vil skrive ut alle partallene mellom 10 og 22. Man kan også bruke *range* og kun angi når løkken skal stoppe, for eksempel når man løper gjennom elementene i en liste. Da kan man skrive: `for i in range(len(liste1)):`. Man dropper altså start og steg parameteren til *range* funksjonen.

En annen ting som er interessant med Listing 8 er bruken av innrykk. Innrykkene i Python har som tidligere nevnt en praktisk funksjon og ikke bare visuell funksjon. Alle etterfølgende kodelinjer som er på samme innrykk som `print(i)` vil være en del av løkken. Definisjonen av løkken er avsluttet ved den første etterfølgende linje befinner seg på samme nivå som `for`.

Et tilsvarende eksempel på konstruksjonen til en *while*-løkke vises i Listing 9.

```
while <en betingelse er evaluert til True>:
    # kodesnutt som skal utføres
```
Listing 9: Konstruksjon til en while-løkke

6.2.4 Egne funksjoner i Python

Når man utfører kommandoer som for eksempel *print* eller *list.append* så kjører man egentlig ferdige funksjoner som ligger i standardbiblioteket til Python. Det er ofte fordelaktig å kunne definere funksjoner selv, spesielt på oppgaver man kanskje ønsker å repetere men der datasettene er forskjellige.

```
# Importerer numpy for å bruke matematiske funksjoner
import numpy as np

# Funksjon som konverterer fra kartesiske til polare koordinater
#   parametere: x og y koordinater.
#               y koordinaten får verdi lik -1 hvis den ikke blir angitt
#   returverdi: liste med r og theta (i radianer)
def polar(x, y=-1):
    r = np.sqrt(x**2 + y**2)
    theta = np.arctan2(y,x)
    return r, theta

# kjører funksjonen med begge parametere
test_polar = polar(-1, 0)
print(test_polar)
# kjører funksjonen med kun x parameter
test_polar = polar(-1)
print(test_polar)
```
Listing 10: Eksempel på definisjon og kjøring av funksjon i Python

Et eksempel på definisjon og bruk av en funksjon er vist i Listing 10. Funksjonen *polar* tar inn kartesiske koordinater og returnerer polare koordinater. Denne funksjonen har samlet to beregninger i en funksjon, og på den måten redusert antall kodelinjer for en bruker hvis denne funksjonen brukes ofte.

En funksjonsdefinisjon starter med nøkkelordet *def*, deretter kommer navnet, og så en eventuell parameterliste til slutt. Man kan sette standard verdi direkte i definisjonen av

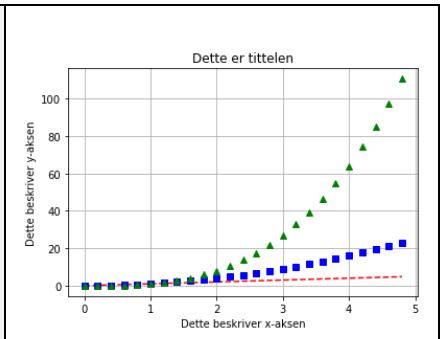
funksjonen (slik det er gjort for y i Listing 10). Da kan denne parameteren utelates når funksjonen kalles.

En funksjon kan også returnere en eller flere verdier, men det er ikke nødvendig. I eksempelet så blir to verdier, r og θ, returnert som en liste. Andre funksjoner kan være laget uten at de returnerer verdier, men for eksempel lager et plott eller skriver til fil eller liknende.

6.3 Plotting med Matplotlib

Matplotlib [7] er et innholdsrikt bibliotek i Python som gir anledning til å visualisere data. Dette kan være en statisk, animert eller til og med ved hjelp av interaktiv visualering. Her vil noen få enkle eksempler bli vist. Det henvises til *matplotlib user's guide* [7] for flere og mer avanserte eksempler.

```python
import matplotlib.pyplot as plt
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.xlabel('Dette beskriver x-aksen')
plt.ylabel('Dette beskriver y-aksen')
plt.title('Dette er tittelen')
plt.grid()
plt.show()
```



Listing 11: Enkelt skript som viser et plot med 3 grafer

Listing 11 er et enkelt eksempel på et skript som plotter tre dataserier som alle er en funksjon av t. De tre funksjonene er $y = t$, $y = t^2$ og $y = t^3$, henholdsvis den røde, blå øog grønne grafen. Dette er alle bestemt av *plt.plot* linjen i skriptet. Det er også lagt til tittel på plott og akser i de påfølgende linjene. *plt.show()* viser plottet. Her kunne man også ha krevet ut plottet til en fil for eksempel.

For å kunne bruke matplotlib er det nødvendig å linke opp biblioteket først med '*import matplotlib.pyplot as plt*'.

```python
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

plt.style.use('ggplot')

font = {'family': 'serif',
        'color':  'black',
        'weight': 'normal',
        'size': 12,
        }

numSamples = 10000
mu, sigma = 100, 15
x = mu + sigma*np.random.randn(numSamples)

# the histogram of the data
numOfBins = 50
n, bins, patches = plt.hist(x, numOfBins, density=True, color='green',
edgecolor='black')
# add a 'best fit' line
y = norm.pdf(bins, mu, sigma)
l = plt.plot(bins, y, 'r--', linewidth=1)
```

```
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'$\mathrm{Histogram\ of\ IQ:}\ \mu=100,\ \sigma=15$', fontdict=font)
plt.axis([40, 160, 0, 0.03])

plt.show()
```

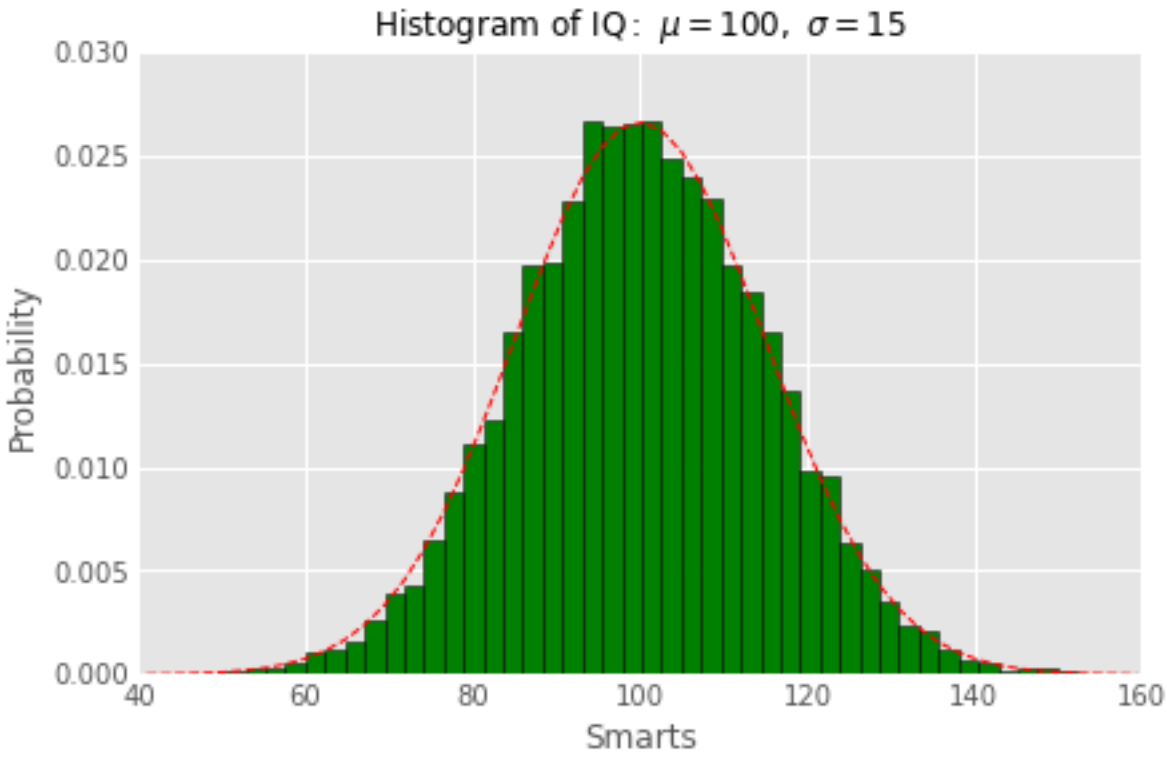
Listing 12: Eksempel som viser plotting av histogram sammen med teoretisk normalfordeling.

Listing 12 er et noe mer avansert eksempel. Her blir 10000 pseudotilfeldige verdier generert med et gjennomsnitt på 100 og standardavvik lik 15. Dette er gjort med funksjonen *np.random.randn*. Deretter blir disse verdiene plottet med *plt.hist* funksjonen. *plt.hist* tar inn med mengde parametere[6], hvor vi i eksempelet bruker noen få:

- x: dataene som skal plottes
- numOfBins: antall bins, altså antall grupper som dataene sorteres i. I dette tilfellet velger vi 50 bins
- density=True: Når *density* er sann betyr det at vi normaliserer histogrammet
- color og edgecolor setter fargene på søylene og på kantene på søylene

Funksjonen returnerer:

- *n:* liste med verdier per bin
- *bins:* liste som gir start og stopp verdi til alle gruppene.
- *patches:* De grafiske objektene som bygger opp histogrammet



Figur 22: Plottet som generert av Listing 12.

[6] Komplett funksjonsdeklarasjon finnes på:
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.hist.html

I tillegg viser eksempelet hvordan man kan plotte en teoretisk normalfordeling basert på dataene i histogrammet. For å generere denne normalfordelingen må funksjonen *norm.pdf* i biblioteket *scipy.stats* brukes. *SciPy* kan man lese mer om på *scipy.org* [8]

Det er også flere visuelle elementer som eksemplifiseres i Listing 12. For eksempel er det mulig å sette en gitt type stil på plottet, det er mulig å velge hvilken teksttype man skal ha, og det er mulig å bruke matematisk språk i titler og tekst. Den resulterende figuren er vist Figur 22.

Det finnes veldig mange flere muligheter i *Matplotlib* enn hva som er skissert her. Det anbefales å oppsøke de offisielle sidene [7].

6.4 PyVISA

PyVISA er et bibliotek i Python for å kontrollere alle slags måleinstrumenter uavhengig av hvilket grensesnitt de har (for eksempel GPIB, RS232, USB, Ethernet). Spesifikasjonen til VISA, eller *Virtual Instrument Software Architecture,* ble definert på midten av 1990-tallet, og det er per i dag implementert på alle større operativsystem. For å bruke Python sammen med VISA, så kaller man funksjoner i et delt VISA bibliotek fra Python.

Det er gitt eksempler på bruk av *PyVISA* for å snakke med Keithley Multimeter i oppgavedokumentet til denne laboratorieøvelsen [9]. Dette finner dere på *mittuib*.

6.5 NI-DAQmx Python API

NI-DAQmx python API [10] kommuniserer med alle National Instruments sine datainnsamlingsenheter, deriblant myDAQ. Siden myDAQ er en av de enklere i National Instruments sin portefølje, så vil NI-DAQmx Python APIet inneholde flere funksjoner som ikke støttes av myDAQen.

6.5.1 Enkelt eksempel på bruk av NI DAQmx

Et enkelt eksempel på kontroll av analog utgang og lesing av analog inngang er gitt i Listing 13. I dette eksempelet er utgang *ao0* koblet direkte til inngang *ai0*. på *myDAQ*. Det vil si at enhver verdi som blir skrevet til *ao0* kan man lese på inngang *ai0*.

Scriptet starter med å importere bibliotekene man behøver, og det viktigste her er *nidaqmx* biblioteket. Deretter genereres det som kalles en *Task* vi har en task *get_ai0_V* som blir knyttet til inngangen *ai0* via kommandoen:

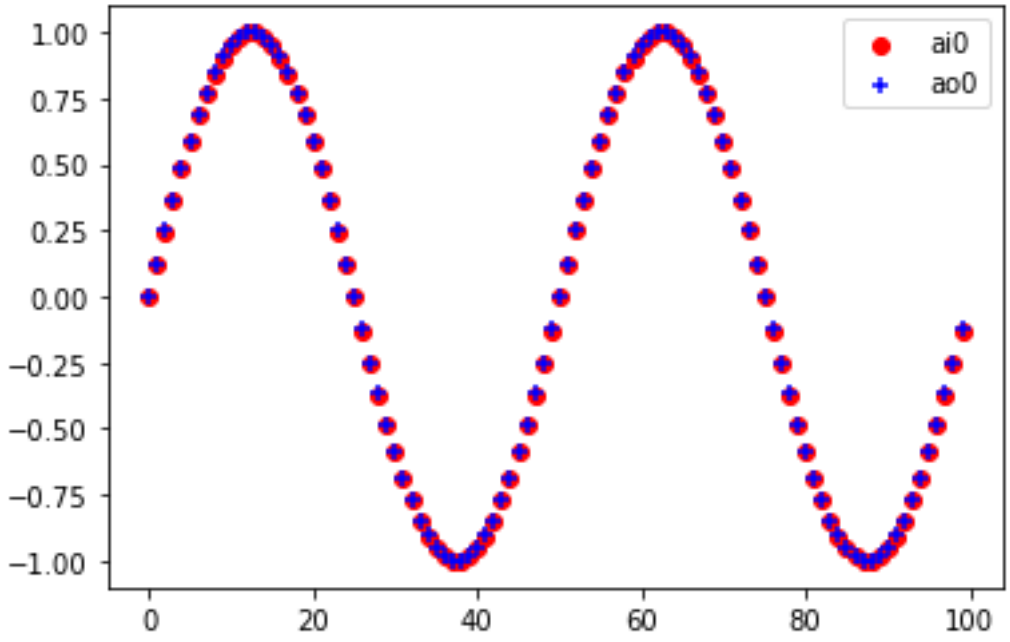
```
get_ai0_V.ai_channels.add_ai_voltage_chan("myDAQ1/ai0")
```

Denne tasken blir opprettet for å kunne lese data fra inngang *ai0*. Tilsvarende lages en task *set_ao_V* for å skrive analog spenning ut på utgangen *ao0*.

Deretter vil *ao0* og *ai0* bli henholdsvis skrevet og lest om en annen i en while-løkke. Til sammen 100 verdier blir skrevet og lest og lagret i en *lister*. Disse verdiene blir så plottet med et scatter plott som vist i Figur 23, der utgangsverdien er gitt i blått og inngangsverdien gitt i rødt.

```
import nidaqmx
import numpy as np
import matplotlib.pyplot as plt

get_ai0_V = nidaqmx.Task()
get_ai0_V.ai_channels.add_ai_voltage_chan("myDAQ1/ai0")
get_ai0_V.start()
set_ao0_V = nidaqmx.Task()
set_ao0_V.ao_channels.add_ao_voltage_chan("myDAQ1/ao0")
set_ao0_V.start()

i=0
x = []
y_ao0 = []
y_ai0 = []

while i<100:
    x.append(i)
    y_val = np.sin(4*np.pi*i/100)
    y_ao0.append(y_val)
    set_ao0_V.write(y_val)
    y_ai0.append(get_ai0_V.read(number_of_samples_per_channel=1))
    i = i+1

plt.scatter(x, y_ai0, c='r', label ="ai0")
plt.scatter(x, y_ao0, c='b', marker="+", label = "ao0")
plt.legend()
plt.show()

get_ai0_V.stop()
get_ai0_V.close()
set_ao0_V.stop()
set_ao0_V.close()
```

Listing 13: Enkelt eksempel som bruker NI-DAQmx biblioteket til å lese og skrive analogverdier på myDAQ



Figur 23: Plott basert på script gitt i Listing 13.

6.5.2 Eksempel med kontrollerbarbar samplingsrate

I forrige eksempel leser vi kun et sample per lesing, og det betyr at sampleraten vår er bestemt av software[7]. Dette medfører at vi får en ustabil og treg samplerate. Hvis man utvider koden i Listing 13 til å også å måle tiden til leseoperasjonen: «`get_ai0_V.read(number_of_samples_per_channel=1)`» så finner en at en lesing i dette eksemplet i gjennomsnitt tar 0.0039 ± 0.0004 sekunder, der usikkerheten er gitt med ett standardavvik. Dette gir en samplingsfrekvens på cirka 260 Hz, mens maksimal samplingsrate til selve myDAQen er på 200 kHz. For å benytte den maksimale samplingsraten må man bruke noe som kalles for *stream_readers*.

Et eksempel på dette er angitt i Listing 14. Her leses 100 samples med en samplingsfrekvens på 20 kHz fra inngang *ai0*. Inngang *ai0* er ikke koblet til en kilde i eksempelet, så derfor ligger inngangen og flyter rundt 0 V. Scriptet starter med å importere biblioteker, og fra *nidaqmx*-biblioteket importeres i tillegg *constants* og *stream_readers*, siden dette er tillegg som er nødvendig for eksempelet.

```
import nidaqmx
from nidaqmx import constants
from nidaqmx import stream_readers
import matplotlib.pyplot as plt
import numpy as np

samplefreq = 20000
numOfSamples = 100

with nidaqmx.Task() as readTask:
    readTask.ai_channels.add_ai_voltage_chan(
        physical_channel="myDAQ1/ai0",
        min_val=-1,
        max_val=1)
    readTask.timing.cfg_samp_clk_timing(
        rate=samplefreq,
        sample_mode=constants.AcquisitionType.CONTINUOUS,
        samps_per_chan=numOfSamples)
    reader = stream_readers.AnalogMultiChannelReader(readTask.in_stream)

    measures = np.zeros([1, numOfSamples])

    readTask.start()
    reader.read_many_sample(
        data = measures,
        number_of_samples_per_channel = numOfSamples)
    readTask.stop()

    measured_time = np.linspace(0, numOfSamples/samplefreq, numOfSamples)

    plt.plot(measured_time, measures[0], 'o', ms=6, label='100 samples')
    plt.xlabel("Time[s]")
    plt.ylabel("Measured Value [V]")
    plt.legend()
    plt.show()
```

Listing 14: Eksempel som sampler ai0 100 ganger med en samplingsfrekvens lik 20kHz.

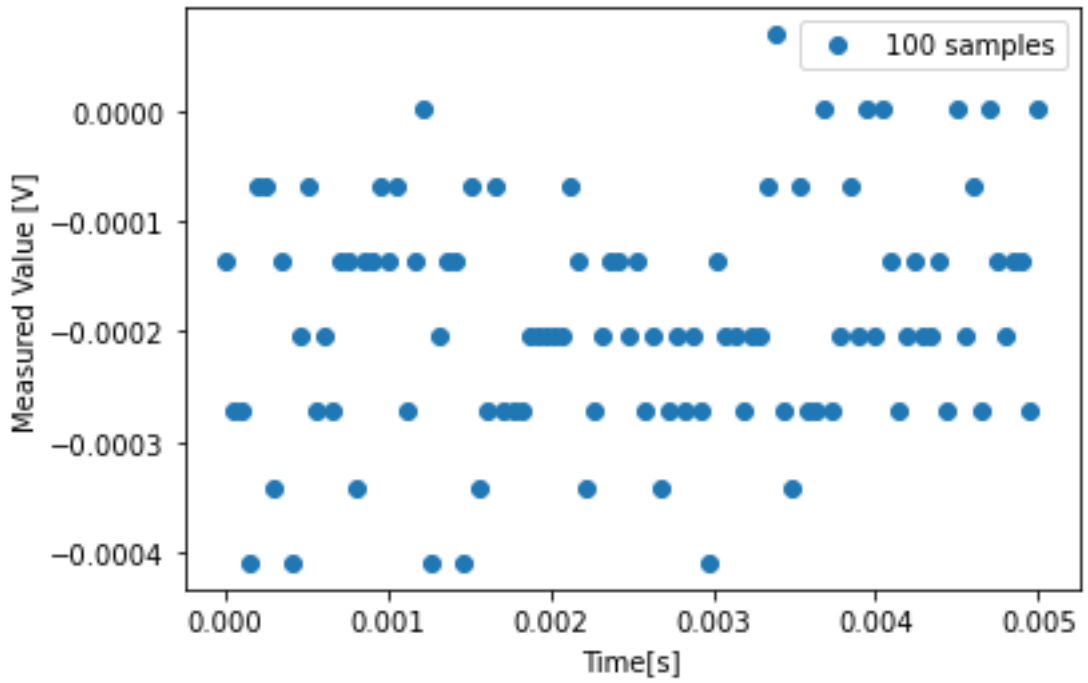
[7] Det svært viktig at man starter og stopper taskene med *start* og *stop* funksjonene. Hvis ikke skjer dette automatisk inni løkken før og etter hver lesing, noe som sinker prosessen betraktelig.

Variablene *samplefreq* og *numOfSamples* setter henholdsvis samplingsfrekvens og antall samples. Det er fornuftig å bruke variabler til dette, siden vi trenger de på flere plasser lenger ned i koden, og da trenger vi bare endre de ett sted om dette er ønskelig.

Kodeeksempelet i Listing 14 instansierer også en *nidaqmx.Task* på en annen måte enn i Listing 13. Her benyttes nøkkelordet *with*[8], som implisitt vil håndtere forskjellige feil med kommunikasjonen til *myDAQ*, og vil fjerne eventuelle pekere i minnet på en forsvarlig måte når programmet avsluttes.

For å lese flere samples om gangen, må først dette konfigureres ved hjelp av funksjonen *readTask.timing.cfg_samp_clk_timing(…)*, og deretter på en *stream_reader* opprettes basert på *in_stream* til *readTask*. Selve samplingen av data skjer ved funksjonen *read_many_sample(…)* som returnerer dataene i parameteren *data*. Dette er et numpy array som må være opprettet før funksjonen kalles.

Til slutt i eksempelet lages en tidsvektor ved hjelp av *linspace*, som starter på tid lik null, og hvor tidshoppene er lik sampleperioden T_s. Deretter blir de 100 samplingene plottet som vist i Figur 24.



Figur 24: Plott basert på scriptet gitt i Listing 14.

For flere gode eksempler til bruk av *NI-DAQmx* biblioteket, samt til Python generelt henvises det til dokumentasjonen fra National Instruments [10], samt Hans-Petter Halvorsen «Python for Control Engineering» [11][9].

[8] Dette er en standard måte å gjøre det på i Python, og er uavhengig av N IDAQmx biblioteket.
[9] På websiden https://halvorsen.blog/documents/programming/python/python.php finnes det flere gode ressurser til Python programmering som er relevant for PHYS114

7 Bibliografi

[1] J. Alme, «PHYS114 Laboratorieoppgave 1 veiledning,» UiB, 2021.

[2] National Instruments, «NI myDAQ User Guide,» [Internett]. Available: https://www.ni.com/pdf/manuals/373060g.pdf. [Funnet 11 08 2020].

[3] National Instruments, «NI myDAQ Device Specifications,» [Internett]. Available: https://www.ni.com/pdf/manuals/373061g.pdf. [Funnet 11 08 2020].

[4] Python Software Foundation, «Python Documentation,» [Internett]. Available: https://docs.python.org/3/contents.html. [Funnet 12 08 2020].

[5] K. Hein, «Introduksjon til Python,» [Internett]. Available: https://krisbhei.github.io/INF2310/Programmering/Python/programmering_python.html. [Funnet 22 08 2020].

[6] The SciPy Community, «NumPy: the absolute basics for beginners,» [Internett]. Available: https://numpy.org/doc/stable/user/absolute_beginners.html. [Funnet 19 08 2020].

[7] The Matplotlib development team, «matplotlib User's Guide,» [Internett]. Available: https://matplotlib.org/users/index.html. [Funnet 24 08 2020].

[8] The SciPy Community, «SciPy Reference Documentation,» [Internett]. Available: https://docs.scipy.org/doc/scipy/reference/. [Funnet 25 08 2020].

[9] J. Alme, «PHYS114 Laboppgave 3 PC-basert datainnsamling,» UiB, Bergen, 2020.

[10] National Instruments, «NI-DAQmx Python API,» [Internett]. Available: https://nidaqmx-python.readthedocs.io/en/latest/index.html. [Funnet 11 08 2020].

[11] H.-P. Halvorsen, «Python for Control Engineering,» 2020. [Internett]. Available: https://halvorsen.blog/documents/programming/python/resources/Python%20for%20Control%20Engineering.pdf. [Funnet 7 10 2020].