07_SEM_EDS_dataprosessering

October 19, 2024

1 Prosessering av Energy Dispersive X-ray Spectroscopy data

Denne Jupyter Notebooken viser hvordan Energy Dispersive X-ray Spectroscopy (EDS) data kan analyseres. Spektroskopi er en veldig viktig datatype "klasse", som dukker opp i mange forskjellige teknikker.

1.0.1 Målet med denne notebooken

- Dere skal kunne prosessere EDS dataene dere tok opp i SEM-laben
- Bli komfortable med å jobbe med spektroskopidata
- Lage figur som viser kjemisk komposisjon i dataene deres

1.0.2 Notebook-planen

- Åpne datasettet, og utforske det
- Finne ut hvilke grunnstoffer som er i prøvematerialet
- Lage figur av dette

1.0.3 Installere exspy

For denne dataøvingen må dere også installere `exspy`, som er et python bibliotek for å analysere spektroskopi-data. Dette må gjøres i virtual environment `pyxem_2024`, i MiniForge:

- `conda install exspy`

De som ikke bruker MiniForge, MiniConda eller Anaconda, kan installere via PyPi hvor pakken heter det samme.

1.1 Importere biblioteker

Først, plotte-biblioteket. Dette kan enten være `%matplotlib qt` for egne vinduer for plottene, eller `%matplotlib widget` for å få plottene i selve Jupyter Notebooken. Så importere HyperSpy

[ ]:
```
%matplotlib qt
import hyperspy.api as hs
```

1.2 Åpne dataset

Dette gjøres via `hs.load`, som kan åpne en rekke dataformater, spesielt innenfor elektronmikroskopi. EDS datasettene fra SEM laben er i en `.hspy` fil.

[ ]:
```
s = hs.load('EDS.hspy')
```

Så kjør selve objektet `s` i en celle for å se på størrelse og dimensjoner.

[ ]:
```
s
```

[ ]:
```
<Signal1D, title: 5c_15kV_3p2nA, dimensions: (2048, 1408|1024)>
```

[ ]:
```
s.set_signal_type('EDS_SEM')      # For EDS in TEM mode, or use 'EDS_SEM' for SEM
```

```
<EDSSEMSpectrum, title: 5c_15kV_3p2nA, dimensions: (2048, 1408|1024)>
```

Her ser vi et par viktige ting: signalet er et `EDSSEMSpectrum`, og den har 3 dimensjoner!

Dimensjonene ser vi helt i slutten, som har 3 tall (som nok vil være annerledes for deres datasett): `(1024, 704|2048)`. Tallene til venstre for `|` er navigasjonsdimensjonene, mens tallet til høyre for `|` er signaldimensjonen: `(NAVIGASJON 0, NAVIGASJON 1|SIGNAL 0)`.

I denne datatypen, så er de 2 navigasjonsdimensjonene probe-posisjonen, og signaldimensjonen er energien til røntgenstrålene.

Dette betyr at signaldimensjonen er 1-dimensjonal, som stemmer bra med at dette er spektroskopisk data.

Av spesiell interesse her er at hver probe-posisjon har ganske lite signal, og består av mange 0. Selve `.hspy` filen er på ca. 13 MB. Sjekk hvor stort datasettet `s` er. Dette gjøres ved å se på `nbytes` attributen, som er i NumPy matrisen i `s` som inneholder selve dataene (`data`). Merk her at dette er i bytes, som betyr at `1000` er en kilobyte, `1000000` er en megabyte, og `1000000000` er en GB.

Dette forteller oss noe om hvor mye effekt vi kan få ut av komprimering av data, spesielt såkalt "sparse" data. Hvor det er veldig lite signal.

[ ]:
```
data_array = s.data  # Henter data fra spektrumobjektet
t = data_array.nbytes  # Finner antall bytes i dataene
print(t)
```

```
2952790016
```

1.3 Enkel utforskning av dataene

Nå kan vi visualisere dataene, og se hvordan spektrumene ser ut. Bruk `plot` funksjonen i `s`, og utforsk datasettet.

Merk: bildene i Jupyter Notebooken kommer til å være forskjellig fra det dere får opp på deres egen datamaskin.

[ ]:
```
s.plot()
```

Her ser vi at hver probe-posisjon har veldig få røntgen-tellinger, ergo at signalet ikke er særlig bra. Noen steder kan vi se at det er klare topper, men disse er for det meste under 10 tellinger.

Dette skal vi gjøre noe med, men først vil vi utforske datasettet som funksjon av røntgenstråle-energien. Bruk transpose funksjonaliteten i `s`, og lag et nytt signal `st`: `st = s.T`

[ ]:
```
st = s.T
```

Skriv `st` i cellen under, og kjør den.

[ ]:
```
st
```

[ ]:
```
<Signal2D, title: 5c_15kV_3p2nA, dimensions: (1024|2048, 1408)>
```

Nå er signalet `Signal2D`: navigasjon- og signal-dimensjonene har blitt byttet om. Så nå kan vi navigere over datasettet som en funksjon av røntgen-energien, istedet for probe-posisjonen.

Så plot `st`. Merk at nå er navigatoren i "røntgen" plottet.

Flytt navigatoren frem og tilbake, spesielt på de klare toppene, og se om forskjelliges steder på prøven lyser opp.

[ ]:
```
st.plot()
```

Selv her, så er tellingene veldig lave. Så la oss midle over flere probe-posisjoner og detektor-kanaler.

For dette, så bruker vi `rebin` funksjonen, som er i `st`. Bruk `scale` parameteren, og sett den til `(4, 8, 8)`, dette betyr at vi summerer 4 detektor-kanaler, 8 x-probe posisjoner, og 8 y-probe posisjoner. Bruk dette til å lage en ny variabel `st_rebin`.

[ ]:
```
st_rebin = st.rebin(scale=(4,8,8))
```

Så kjør `st_rebin`, for å se dimensjonene til dette nye signalet.

[ ]:
```
st_rebin
```

[ ]:
```
<Signal2D, title: 5c_15kV_3p2nA, dimensions: (256|256, 176)>
```

Her ser vi at sammenlignet med det orginale `st` signalet, som var `(2048|1024, 704)`, så er `st_rebin` mye mindre. Spesifikt, at den nye størrelsen er `2048 / 4 | 1024 / 8, 704 / 8)`.

Deretter plot dette nye `st_rebin` signalet.

[ ]:
```
st_rebin.plot()
```

Nå er røntgen-tellingene mye høyere.

Så vi ser at det er noe interessant i dataene. Det neste steget er å lage bilder som viser hvor de forskjellige grunnstoffene er.

1.4 Mer avansert

1.4.1 Finne grunnstoffene

Det første vi må gjøre her, er å finne ut hvilke grunnstoffer vi har i prøvematerialet.

Enkleste måten å gjøre dette på, er å se på toppene vi har røntgen-signalet, kombinert med det vi vet om prøvematerialet.

Så: summer opp alle probeposisjonene, til et røntgen-energi signal. Bruk `sum` funksjonen i `s` til å lage et nytt signal `s_sum`.

[ ]:
```
s_sum = s.sum()
```

Så bruk `plot` funksjonen i `s_sum` til å visualisere dette signal, og finn ut hva slags grunnstoffer vi har.

Dere kan se hvilke topper de forskjellige grunnstoffene lager, ved for eksempel å bruke denne: https://www.jeolusa.com/Portals/2/brochures/JEOL%20EDS%20Periodic%20Table.jpg?ver=CI-OGZn69_wK8jSagOFuKw%3d%3d

1; 0.050 Ukjent 2; 0.273 C Carbon 3; 0.390 N Nitrogen 4; 0.521 O Oksygen 5; 0.722 Fe Iron 6; 0.852 Ni Nickel 7; 1.100 Ga Gallium 8; 1.495 Al Aluminium 9; 1.741 Si Silicium 10; 6.390 Fe Iron 11; 7.475 Ni Nickel

[ ]:
```
s_sum.plot()
```

Gå igjennom alle toppene, og prøv å finn ut hvilket grunnstoff de tilhører.

Merk at samme grunnstoff kan ha flere topper, så hvis dere er usikre så er et triks å sjekke om de andre toppene også er med spektrumet.

1.5 Kalibrering av data

Et viktig aspekt med alle typer data, er at de er kalibrert riktig. For SEM-EDS data, så betyr det både de romlige dimensjonene (probe-posisjonene), og energien til røntgenstrålene. Dere kan IKKE anta at det som automatisk kommer fra instrumentene er riktig.

Måten man gjør dette på er å bruke en kjent størrelse til å kalibrere dataene. For bildedata er dette ofte et eller annet objekt, mens for EDS data så bruker man topper fra noen kjente grunnstoffer.

Bruk grunnstoffene du fant til å kalibrere energi-aksen til `s_sum` signalet. Bruk `calibrate` funksjonen som er i `s_sum`. Da vil du få opp et plot av `s_sum`, og bruk venstre-klikk + dra i dette plottet for å markere en kjent distanse. For eksempel mellom to topper de vet hva er.

[ ]:
```
s_sum.calibrate()
```

```
VBox(children=(HBox(children=(FloatText(value=0.0, description='New left'),␣
↪Label(value='keV', layout=Layout(w…
```

Så overfør dette til det orignale `s` signalet

[ ]:
```
s.axes_manager[-1].scale = s_sum.axes_manager[-1].scale
s.axes_manager[-1].offset = s_sum.axes_manager[-1].offset
```

1.5.1 Rebinning av signalet

Som vi så tidligere, så er det litt få signaler i hver probe-posisjon. Så før vi begynner med den mer avanserte prosesseringen, så bruk `rebin` funksjonen i `s`, og bruk `scale` parameteren med `(8, 8, 1)` til å summere 8 x 8 probe-posisjoner. Bruk dette til å lage en ny variabel, `s_rebin`.

Hvis du er usikker på hvordan du gjør dette, husk at du kan få opp `docstring` for alle funksjoner i python ved å ha en `?` etter funksjonen. Så her, `s.rebin?`.

[ ]:
```
s_rebin = s.rebin(scale=(8,8,1))
```

1.5.2 Legge til grunnstoffene i signalet

Nå som dere har funnet grunnstoffene, så må de legges til i `s_rebin` signalet.

Dette gjøres via `set_elements` funksjonen som er i `s_rebin`. Parameter som skal til `set_elements` må være en liste, og hvert grunnstoff må være i formen `"Si"`, `"Fe"`, ...

Tips: se i docstring til `set_elements` via "Shift + Tab" på tastaturet.

[ ]:
```
s_rebin.set_elements(['C','N','O','Ga','Al','Si','Fe','Ni'])
```

Sjekk at alt har blitt lagt til, via `metadata.Sample` attribute i `s_rebin`

[ ]:
```
s_rebin.metadata
```

[ ]:
```
    Acquisition_instrument
        SEM
            Detector
                EDS
                    azimuth_angle = 0.0
                    elevation_angle = 35.0
                    energy_resolution_MnKa = 130.0
            Stage
                tilt_alpha = 0.0
    EDS
        Accelerating Voltage = 15,00kV
        Azimuth (degrees) = 0,0
        Created = 19.09.2024 14:06:26
        Detector Type = X-Max
        Detector Type Id = 29
        Elevation (degrees) = 35,0
        Energy Range (keV) = 10 keV
        Energy per Channel (eV) = 10,0eV
        Label = Map Sum Spectrum
        Livetime = 115,3s
        Magnification = 29363 x
        Number Of Channels = 1024
        Primary Detector =
        Primary Detector Serial Number = 75099
        Process Time = 4
        Pulse Pile Up Correction = Succeeded
        Source = Acquired
        Specimen Tilt (degrees) = 0,0
        Window Type = SATW
        Working Distance = 3,9mm
    General
        FileIO
            0
                hyperspy_version = 2.1.0
```

```
            io_plugin = rsciio.ripple
            operation = load
            timestamp = 2024-09-28T15:12:03.717242+02:00
        1
            hyperspy_version = 2.1.0
            io_plugin = rsciio.hspy
            operation = save
            timestamp = 2024-09-28T15:12:03.757334+02:00
        2
            hyperspy_version = 2.1.1
            io_plugin = rsciio.hspy
            operation = load
            timestamp = 2024-10-19T20:48:19.751263+02:00
    date =
    original_filename = EDS Data 2.rpl
    time =
    title = 5c_15kV_3p2nA
Sample
    elements = ['Al', 'C', 'Fe', 'Ga', 'N', 'Ni', 'O', 'Si']
Signal
    signal_type = EDS_SEM
```

Så legger vi til røntgen linjene til disse grunnstoffene, via `add_lines` funksjonen i `s_rebin`

[ ]:
```
s_rebin.add_lines()
```

Så se hva som har blitt lagt til i metadataen, via `metadata.Sample` attribute i `s_rebin`

[ ]:
```
s_rebin.metadata
```

[ ]:
```
Acquisition_instrument
    SEM
        Detector
            EDS
                azimuth_angle = 0.0
                elevation_angle = 35.0
                energy_resolution_MnKa = 130.0
        Stage
            tilt_alpha = 0.0
EDS
    Accelerating Voltage = 15,00kV
    Azimuth (degrees) = 0,0
    Created = 19.09.2024 14:06:26
    Detector Type = X-Max
    Detector Type Id = 29
    Elevation (degrees) = 35,0
    Energy Range (keV) = 10 keV
    Energy per Channel (eV) = 10,0eV
```

```
        Label = Map Sum Spectrum
        Livetime = 115,3s
        Magnification = 29363 x
        Number Of Channels = 1024
        Primary Detector =
        Primary Detector Serial Number = 75099
        Process Time = 4
        Pulse Pile Up Correction = Succeeded
        Source = Acquired
        Specimen Tilt (degrees) = 0,0
        Window Type = SATW
        Working Distance = 3,9mm
    General
        FileIO
            0
                hyperspy_version = 2.1.0
                io_plugin = rsciio.ripple
                operation = load
                timestamp = 2024-09-28T15:12:03.717242+02:00
            1
                hyperspy_version = 2.1.0
                io_plugin = rsciio.hspy
                operation = save
                timestamp = 2024-09-28T15:12:03.757334+02:00
            2
                hyperspy_version = 2.1.1
                io_plugin = rsciio.hspy
                operation = load
                timestamp = 2024-10-19T20:48:19.751263+02:00
        date =
        original_filename = EDS Data 2.rpl
        time =
        title = 5c_15kV_3p2nA
    Sample
        elements = ['Al', 'C', 'Fe', 'Ga', 'N', 'Ni', 'O', 'Si']
        xray_lines = ['Al_Ka', 'C_Ka', 'Fe_La', 'Ga_La', 'N_Ka', 'Ni_La',
 'O_Ka', 'Si_Ka']
    Signal
        signal_type = EDS_SEM
```

Her ser dere at det bare har blitt lagt til en linje per grunnstoff. Dette er fordi den med lavest energi er det mest relevant.

Dette kan dere så plotte, ved å bruke `plot` med `xray_lines=True` argumentet

[ ]:
```
s_rebin.plot(xray_lines=True)
```

Så kan vi hente ut intensiteten for alle disse linjene, over hele datasettet.

Til dette bruker vi `get_lines_intensity` som er i `s_rebin`. Lagre resultatet til dette i en ny variabel: `linjer`. I tillegg, så bruk `plot_result=True` i `get_lines_intensity`. Dette vil åpne en plotte-vindu for hvert grunnstoff.

[ ]:
```
linjer = s_rebin.get_lines_intensity()#plot_result=True)
```

Så kan vi se hva som er i `linjer` variablen. Skriv `linjer` i cellen under, og kjør den.

[ ]:
```
linjer
```

[ ]:
```
[<BaseSignal, title: X-ray line intensity of 5c_15kV_3p2nA: Al_Ka at 1.49 keV,
dimensions: (256, 176|)>,
 <BaseSignal, title: X-ray line intensity of 5c_15kV_3p2nA: C_Ka at 0.28 keV,
dimensions: (256, 176|)>,
 <BaseSignal, title: X-ray line intensity of 5c_15kV_3p2nA: Fe_La at 0.70 keV,
dimensions: (256, 176|)>,
 <BaseSignal, title: X-ray line intensity of 5c_15kV_3p2nA: Ga_La at 1.10 keV,
dimensions: (256, 176|)>,
 <BaseSignal, title: X-ray line intensity of 5c_15kV_3p2nA: N_Ka at 0.39 keV,
dimensions: (256, 176|)>,
 <BaseSignal, title: X-ray line intensity of 5c_15kV_3p2nA: Ni_La at 0.85 keV,
dimensions: (256, 176|)>,
 <BaseSignal, title: X-ray line intensity of 5c_15kV_3p2nA: O_Ka at 0.52 keV,
dimensions: (256, 176|)>,
 <BaseSignal, title: X-ray line intensity of 5c_15kV_3p2nA: Si_Ka at 1.74 keV,
dimensions: (256, 176|)>]
```

Sjekke integrasjonsvinduet Her ser vi at den er en liste med signaler, et for hvert grunnstoff. For å plotte dem: `linjer[0].plot()`

Et mulig problem med denne typen prosessering, er hvis røntgen-linjene er så nærme at de overlapper. En måte å sjekke dette på, er å se hvor "bredde" integrasjonsvinduene er.

Dette gjøres enkelts med å først å summere datasettet igjen. Bruk `sum` via `s_rebin`, til å lage en ny variabel `s_sum2`.

[ ]:
```
linjer[0].plot()
s_sum2 = s_rebin.sum()
```

Så `plot` `s_sum2`, med argumentet `integration_windows='auto'`. Bruk forstørrelse-funksjonen til å sjekke at det ikke er for mye overlapp.

[ ]:
```
s_sum2.plot(integration_windows='auto')
```

1.5.3 Lage bilder av hvor grunnstoffene er

Nå som vi kan se hvor de forskjellige grunnstoffene er, så lager vi en figur som viser dette.

Først henter vi ut et og et grunnstoff, kall disse `s_si`, `s_fe`, … . Siden `linjer` er en liste, så gjøres dette med `linjer[0]`, `linjer[1]`, … . Pass på å sjekke hvilket grunnstoff de forskjellige signalene er!

Viktig: disse signalene må transposes! Så kommandoen blir `linjer[0].T`.

[ ]:
```
s_Al = linjer[0].T
s_C = linjer[1].T
s_Fe = linjer[2].T
s_Ga = linjer[3].T
s_N = linjer[4].T
s_Ni = linjer[5].T
s_O = linjer[6].T
s_Si = linjer[7].T
```

Så kan vi lage en matplotlib figur med et subplot per grunnstoffer dere har.

- Tips 1: har dere backscatter elektron eller Sekundærelektron bilde, så er det også fint å ta med! Da må dere ha et ekstra subplot i tillegg.
- Tips 2: hvis dere har veldig mange grunnstoff, så kan den være en fordel å ha 2 vertikale rader med subplot.
- Tips 3: bruk figsize parameteren til å lage figures større, slik at subplotene passer inn. F.eks. hvis dere har 4 horisontale subplot, så kan `figsize=(20, 5)` passe bra.

Importer matplotlib

[ ]:
```
import matplotlib.pyplot as plt
```

Bruk `subplot_mosaic` i `plt` til å lage en figur med 2 rader med subplots. Første definer geometrien `mosaic`:

```
mosaic = """
    abc
    de.
    """
```

Merk at `.` betyr at det ikke kommer noe plot der. Test f.eks. med `d.e`. Så lag `fig` og `ax_dict` med `plt.subplot_mosaic`, ved å bruke `mosaic` og `figsize=(9, 4))`

[ ]:
```
mosaic = """
abcd
efgh
"""
fig, ax_dict = plt.subplot_mosaic(mosaic, figsize=(15,10))
```

Så hent ut `ax_..` objekter. Husk at `ax_dict` er en "dictionary"

[ ]:
```
ax_Al = ax_dict['a']
ax_C = ax_dict['b']
ax_Fe = ax_dict['c']
ax_Ga = ax_dict['d']
ax_N = ax_dict['e']
ax_Ni = ax_dict['f']
ax_O = ax_dict['g']
ax_Si = ax_dict['h']
```

Så bruk `imshow` med forskjellige fargekart (`cmap`) til å visualisere dataene: for eksempel `"Blues_r"`, `"Greens_r"`, … Se matplotlib dokumentasjonen for en fullstendig liste over alle fargekartene.

Et ekstra plotte-element som burde være med her er en ting som viser antall tellinger. Dette gjøres ved å:

- `cax_si = ax_si.imshow(....., extent=s_si.axes_manager.signal_extent, ....)`

[ ]:
```
cax_Al = ax_Al.imshow(s_Al,cmap='Blues_r',extent=s_Al.axes_manager.
 ↪signal_extent)
cax_C = ax_C.imshow(s_C,cmap='Greens_r',extent=s_C.axes_manager.signal_extent)
cax_Fe = ax_Fe.imshow(s_Fe,cmap='BrBG',extent=s_Fe.axes_manager.signal_extent)
cax_Ga = ax_Ga.imshow(s_Ga,cmap='turbo',extent=s_Ga.axes_manager.signal_extent)
cax_N = ax_N.imshow(s_N,cmap='plasma',extent=s_N.axes_manager.signal_extent)
cax_Ni = ax_Ni.imshow(s_Ni,cmap='inferno',extent=s_Ni.axes_manager.
 ↪signal_extent)
cax_O = ax_O.imshow(s_O,cmap='GnBu',extent=s_O.axes_manager.signal_extent)
cax_Si = ax_Si.imshow(s_Si,cmap='CMRmap',extent=s_Si.axes_manager.signal_extent)
```

1.5.4 Sette clim

I noen datasett, så kan det være vanskelig å se de interessante delene av dataene, fordi andre deler av datasettet entent har veldig høye eller veldig lave verdier.

Hvis dette er tilfellet, så bruk `cax_...` variablene til å sette "fargenivået" i plottene. Dette gjøres via `set_clim` funksjonen i (f.eks.) `cax_si`.

Merk at dere må finne ut hva de gode verdiene er her. En måte å gjøre det er å plotte dataene (f.eks. i `s_si`), flytte musepekeren over de interessante områdene, og se hva verdien er der (helt øverst til høyre `intensity`).

[ ]:
```
#plt.plot(s_Al)
cax_Al.set_clim()
cax_C.set_clim()
cax_Fe.set_clim()
cax_Ga.set_clim()
cax_N.set_clim()
cax_Ni.set_clim()
cax_O.set_clim()
cax_Si.set_clim()
```

Legge til colorbars I denne typen figurer, så er det fint å vise hvor mange tellinger man har. Dette gjøres via `colorbar`.

- `fig.colorbar(cax_si, ax=ax_si, label="Si")`

Merk: hvis dere får to colorbars på samme sublplot, så har dere kjørt `fig.colorbar(...)` to ganger på samme subplot. For å fikse dette, så lag figuren på nytt, fra `plt.subplots(...)`.

[ ]:
```
ori ='horizontal'
padd=0.02
fig.colorbar(cax_Al, ax=ax_Al, label='Al',orientation=ori,pad=padd)
fig.colorbar(cax_C, ax=ax_C, label='C',orientation=ori,pad=padd)
fig.colorbar(cax_Fe, ax=ax_Fe, label='Fe',orientation=ori,pad=padd)
fig.colorbar(cax_Ga, ax=ax_Ga, label='Ga',orientation=ori,pad=padd)
fig.colorbar(cax_N, ax=ax_N, label='N',orientation=ori,pad=padd)
fig.colorbar(cax_Ni, ax=ax_Ni, label='Ni',orientation=ori,pad=padd)
fig.colorbar(cax_O, ax=ax_O, label='O',orientation=ori,pad=padd)
fig.colorbar(cax_Si, ax=ax_Si, label='Si',orientation=ori,pad=padd)
```

[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ae5f2ed0>
```

[ ]:
```
# Assuming ax_Al is part of a subplot grid
for ax, label in zip([ax_Al, ax_C, ax_Fe, ax_Ga,ax_N, ax_Ni, ax_O,␣
 ↪ax_Si],['A)', 'B)', 'C)', 'D)', 'E)', 'F)', 'G)', 'H)']):
    ax.set_xticks([])
    ax.set_yticks([])
    ax.annotate(label, xy=(0.03, 0.88), xycoords='axes fraction',
                fontsize=12, color='white', fontweight='bold',  # Make text␣
 ↪bold and white
                bbox=dict(facecolor='black', alpha=0.7, edgecolor='none'))  #␣
 ↪Add black background with some transparency

fig.subplots_adjust(hspace=0,left=0.01,right=0.99)

fig.savefig('SEM EDS_5c_15kV_3p2nA.png',bbox_inches='tight')#pad_inches=0
```

Så bruk tingene dere har lært i de andre dataøvingen, til å lage en figur:

- Legg til scalebar
- Ha annoteringer (a, b, c, …), skriv hvilket grunnstoff det er: “Fe”, “Si”, …
- Fjern tomrommet som kommer rundt plottene
- Fjern tallene som er rundt bildene. Her må dere mest sannsynlig manuelt stille på `figsize` til det blir bra

Eksempel på en sånn figur, men uten alle annoteringene