04_TEM_STEM_bildedata

October 24, 2024

1 Plotting av TEM bildedata

Målet med denne Jupyter Notebooken er at dere skal bli litt kjent med HyperSpy, og hvordan dere kan både åpne og visualisere TEM data. Disse er vanligvis i DM3-formatet, hvor HyperSpy kan lese både dataen og metadataen.

Denne kunnskapen blir nyttig i neste Notebook, hvor dere skal prosessere mer avansert data: STEM - Differential Phase Constrast (DPC) og Scanning Electron Diffraction (SED). I tillegg skal HyperSpy sin spektroskopi-funksjonalitet brukes i neste øving, til å analysere data fram SEMen.

1.1 Instruksjoner

Før dere begynner med denne Jupyter Notebooken er det viktig at dere har gått igjennom instruksjonene:

- Rigge opp "pyxem" environment, som har alle python pakkene dere trenger: `pyxem`, `hyperspy`, `hyperspy-gui-ipywidgets`, `hyperspy-gui-traitsui`, `ipympl` og `jupyterlab`
- Aktivere dette "pyxem" environment
- Ha dataene i samme mappe som Jupyter Notebookene

1.2 Blackboard levering

Fra denne Jupyter Notebooken skal dere levere både bildefil og HTML-versjon av Jupyter Notebooken dere brukte til å lage bildefilen. HTML-filen må være i en ZIP-fil.

Eksempel på hvordan bildet kan se ut, med skala-indikator, "høy-kontrast" tekst, og litt annoteringer. Merk at dere skal ikke bruke akkurat de annoteringene: bildet er bare en indikator for hvordan denne typen figurer burde se ut.

Merk at dere skal OGSÅ levere bildefil og HTML-versjon av Jupyter Notebooken for enten STEM-DPC eller SED: se de andre Notebooken (05 og 06) for mer informasjon om dette.

1.3 Importere plotte-bibliotek og `hyperspy`

Først må plotte-biblioteket defineres, med `%matplotlib qt`, eller `%matplotlib widget`. Jeg anbefaler `%matplotlib qt`. Av og til så virker den ikke, i dette tilfellet, bruk `widget`.

[ ]:
```
%matplotlib qt
```

Så importer `hyperspy.api` som `hs`.

Dere kan få en `WARNING:hyperspy_gui_...`. Denne kan ignoreres.

[ ]:
```
import hyperspy.api as hs
```

1.4 Åpne dataset

Dette gjøres via `hs.load`, som kan åpne en rekke dataformater, spesielt innenfor elektronmikroskopi.

Velg en av datasettene deres som har filformat `.dm3`. Gjerne en av de som dere kan tenke å bruke i labrapporten deres.

Lag et objekt som heter `s`.

Tips: sjekk docstring for informasjon om hvordan `hs.load` virker.

[ ]:
```
s = hs.load('/Users/toham1999/Koding/24H 5.semester/Nanoverktøy/Dataøving_3_TEM/
↪dataøving_3_tem_eks/datasett/annular_dark_field_data.dm3')
```

Dette lager et `Signal2D` objekt, som inneholder mange forskjellige funksjoner.

En av disse er en plotte-funksjon, som visualiserer dataene.

Bruk `plot` funksjonen, som er en del av `s` objektet. Hvis du har brukt `%matplotlib qt` så åpnes dette som et eget vindu, dette kommer ofte "bak" nettleseren denne Notebooken er i. Så hvis vinduet ikke blir synlig, prøv å minimer nettleseren.

Hvis du bruker `%matplotlib widget` så kommer plottet rett i notebooken.

[ ]:
```
s
```

[ ]:
```
<Signal2D, title: JEOL BF _ ADF_x300k, dimensions: (|1024, 1024)>
```

Dette er et interaktivt plot. Merk at dataene er automatisk kalibrert, ved at det er en scalebar med enten nano eller mikrometer.

I tillegg kan kontrasten endres med kontrast editoren: trykk på bildet, og så trykk på `H` knappen. Merk at dette bare virker med `%matplotlib qt`. Hvis du må bruke `%matplotlib widget`, så kan dette endres med `s.plot(vmin=0, vmax=1000)`.

Dette endrer bare visualiseringen i plottet, ikke dataene. Her er dette bare for å vise funksjonalitetene, dere trenger ikke å gjøre noe spesielt med dette plottet.

1.5 `Signal2D` strukturen

Denne måten å jobbe med data er veldig praktisk, siden funksjonene finnes i data-objektet (signalet). Selve dataene er en NumPy array.

Skriv `s.data`, og kjør cellen.

[ ]:
```
s.data
```

[ ]:
```
array([[674183, 639611, 667666, …, 750693, 712613, 761718],
       [655082, 642951, 616608, …, 760770, 718168, 769597],
       [680179, 607293, 630039, …, 716128, 718371, 703630],
       …,
```

```
       [     0,      0,      0, …, 778246, 830771, 824499],
       [     0,      0,      0, …, 792662, 787962, 824042],
       [     0,      0,      0, …, 834342, 898970, 857274]],
      dtype=uint32)
```

Her kan vi se alle tall-verdiene til dataene i plottet vi så tidligere.

Men dette er bare selve dataene, metadatanene sånn som kablibrering er i `s.axes_manager`. Prøv dette.

[ ]:
```
s.axes_manager
```

[ ]:
```
<Axes manager, axes: (|1024, 1024)>
            Name |   size |  index |  offset |   scale |  units
================ | ====== | ====== | ======= | ======= | ======
---------------- | ------ | ------ | ------- | ------- | ------
               x |   1024 |      0 |      -0 |    0.49 |     nm
               y |   1024 |      0 |      -0 |    0.49 |     nm
```

Disse tallene er vanligvis riktige, men alltid sjekk at de ser rimlige ut! For "standard" moduser så pleier de å være greie, men f.eks. i STEM-DPC data så er det mest sannsylig feil. Den vanligste måten å sjekke disse kalibreringene, er via objekter med kjente størrelser. For eksempel størrelsen på "vinduene", eller hvis man har atomæroppløsning så kan man bruke atomgitteret.

For å kalibrere dataene, så kan dere bruke `s.calibrate` funksjonen. Denne åpner opp et plottevindu av signalet, og en interaktiv linje.

[ ]:
```
#s.calibrate()
```

1.5.1 Metadata

Andre metadata finner dere i `s.metadata`. Finn ut hva akselerasjonsspenningen (beam energy) var, via `Acquisition_instrument` - `TEM` (Trykk på pilen)

[ ]:
```
s.metadata
```

[ ]:
```
    Acquisition_instrument
        TEM
            acquisition_mode = STEM
            beam_current = 0.0
            beam_energy = 200.0
            camera_length = 200.0
            dwell_time = 2e-05
            magnification = 300000.0
            microscope = 2100F
    General
        FileIO
            0
                hyperspy_version = 2.1.1
                io_plugin = rsciio.digitalmicrograph
```

```
            operation = load
            timestamp = 2024-10-24T18:59:17.338568+02:00
        authors = Emil Christiansen
        date = 2021-03-07
        original_filename = annular_dark_field_data.dm3
        time = 12:15:17
        title = JEOL BF _ ADF_x300k
    Sample
        description = TEDPELLA673_500nmGrid
    Signal
        Noise_properties
            Variance_linear_model
                gain_factor = 1.0
                gain_offset = 0.0
        quantity = Intensity
        signal_type =
```

Her finner man informasjon om mikroskopet, sånn som akselerasjonspenning. Informasjon om prøven er ting man vanligvis må manuelt skrive inn, så denne informasjon er ganske ofte feil.

1.5.2 Måle avstander

En viktig del av mikroskopi-data, er å måle størrelse på objekter. Først, kjør cellen under.

[ ]:
```python
def get_line_profile(s):
    roi = hs.roi.Line2DROI(linewidth=0.5)
    s.plot()
    s_roi = roi.interactive(s)
    s_roi.plot(autoscale="xyv")
    roi_span = hs.roi.SpanROI()
    roi_span.interactive(s_roi)
    return roi_span
```

Deretter bruk `s` variablen som parameter inn i `get_line_profile`, og lagre resultatet av dette som `roi_span`.

`roi_span` har to "attributes", som inneholder venstre og høyre posisjonen til "span" i linjeprofil-plottet. Disse er `left` og `right`.

Regn ut lengden ved å trekke disse fra hverandre, og lagre det som ny variabel `span_length`.

[ ]:
```python
roi_span = get_line_profile(s)
```

1.6 Plotting av data

Laging av bildefiler er veldig likt som for FIB-bildene, men det er et par triks.

Først importer `matplotlib.pyplot` som `plt`, og lag en `fig` og `ax` via `plt.subplots`. Deretter bruk `imshow` i `ax`, med `s.data` som bildedata, og `s.axes_manager.signal_extent` som `extent`.

[ ]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(6,5))
ax.imshow(s.data, extent=s.axes_manager.signal_extent)
```

[ ]:
```
<matplotlib.image.AxesImage at 0x181a69330>
```

Så kan man bruke akkurat de samme plotte-funksjonene som med FIB-dataene.

1.6.1 Endre på kontrast: `clim`

Et vanlig problem i denne type data er store forskjeller i intensitet i bildet. Ergo at det er veldig lyse og veldig mørke området, som gjør det vanskelig å se detaljene man er interessert i.

En løsning på dette er å lagre `AxesImage` som en variabel (`cax = ax.imshow(.....)`, og bruke `set_clim` funksjonen i `cax`.

Tips - Bruk `get_clim` for å se hva de automatiske verdiene er, og bruk disse som et utgangspunkt - Sjekk docstring til `set_clim` for å se hvordan den virker (`Shift` + `Tab`)

[ ]:
```
cax = ax.imshow(s.data, extent=s.axes_manager.signal_extent,cmap='turbo')
cax.set_clim()
ori ='vertical'
padd=0.02
fig.colorbar(cax, ax=ax, label='Grain orientation',orientation=ori,pad=padd)
ax.set_xticks([]),ax.set_yticks([])
ax.annotate(text='A', xy=(0.03, 0.93), xycoords='axes fraction',fontsize=14,␣
 ↪color='white', fontweight='bold')


from mpl_toolkits.axes_grid1.anchored_artists import AnchoredSizeBar
import matplotlib.font_manager as fm
import matplotlib.patheffects as patheffects
fontprops = fm.FontProperties(size=14)
scalebar_kwargs = {'size': 100, 'label': '100 nm', 'loc': 4, 'frameon': False,␣
 ↪'color': 'white', 'size_vertical': 10, 'label_top': False, 'fontproperties':␣
 ↪fontprops}
scalebar = AnchoredSizeBar(transform=ax.transData, **scalebar_kwargs)
# Denne legger til et svart omriss rundt scalebar teksten, for å gjøre den␣
 ↪lettere å lese
scalebar.txt_label._text.set_path_effects([patheffects.withStroke(linewidth=2,␣
 ↪foreground='black', capstyle="round")])
ax.add_artist(scalebar)


fig.savefig('04_TEM.png',bbox_inches='tight',dpi=300)
```

2 TEM-data-figur

Bruk denne Notebooken i tillegg til kunnskapen og koden dere brukte i FIB-øvingen til å lage en bildefil med TEM-data, gjerne en dere tenker å bruke i labrapporten.

Denne skal ha samme "format" som FIB-bildet:

- Inneholde et eller flere bilder
- Ha scalebar
- Noen annoteringer

[ ]:

3 Neste Jupyter Notebook: STEM-DPC eller SED

Nå som dere kan basisen om hvordan HyperSpy virker, så fortsett med enten `..._STEM_DPC...` eller `..._scanning_electron_diffraction_...` notebooken. Dere skal bare levere en av disse på blackboard, men hvis dere har begge datatypene fra TEM-laben så anbefaler jeg å gå igjennom begge.