05_STEM_DPC_dataprosessering

October 24, 2024

1 Prosessering av STEM-DPC data

Denne Jupyter Notebooken viser hvordan Scanning Transmission Electron Microscopy - Differential Phase Contrast (STEM-DPC) data kan analyseres. Sammenlignet med analyse av "standard" TEM data som dere så på i forrige Notebook, så er dette mer komplisert på grunn av datastørrelsen: det er veldig enkelt å gå tom for minne, noe som (mest sannsynlig) gjør at datamaskinen deres kræsjer.

1.0.1 Målet med denne notebooken

- Dere skal kunne prosessere de magnetiske STEM-DPC dataene deres fra TEM-laben
- Bli komfortable med å jobbe med 4-dimensjonelle datasett
- Lære litt enkle verktøy og strategier for å jobbe med store datasett, som ofte er mye større en tilgjengelig minne

1.0.2 Notebook-planen

- "Åpne" datasettet uten å laste det inn i minnet, "lazily"
- Utforske datasettet, uten å laste det inn i minnet. Via "lazy plotting"
- Redusere datamengden, slik at vi kan laste det inn i minnet
- Bruk center of mass til å finne den magnetiske vektoren i hver probe-posisjon
- Visualisere den magnetiske domenestrukturen i en bildefil

Eksempel på bilde:

Selve datasettene dere skal se på her er på ca. 8 GB, noe som er ganske smått i "4-D STEM" verdenen: disse kan lett være 100+ GB. Så selv om dere har en datamaskin som takler 8 GB, så anbefaler jeg at dere følger prosedyren for å redusere datastørrelsen.

Merk: for å gjøre denne dataøvingen, så må dere ha lastet ned datasettet `stem_dpc_data.hspy` og legge det i `datasett` mappa: https://filesender.sikt.no/?s=download&token=81ef6a80-1738-40c5-bc19-a05c48aa83d0 (denne lenken vil være tilgjengelig fram til 20. november)

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.

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

Så importere HyperSpy

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

2 Konvertere datasett

Det første vi må gjøre er å konvertere dataene til `.zspy` formatet, dette for å kunne prosessere dataene mye raskere. Hvis dere jobber med eget datasett, så må dere endre på filnavn i begge cellene under.

Merk: dette trenger bare å gjøres en gang, når dere har lagd `.zspy` filen så trenger dere ikke å gjøre det igjen.

[ ]:
```
#s_dpc = hs.load("datasett/stem_dpc_data.hspy", lazy=True)
#s_dpc = hs.load("datasett/test_merlin.hspy", lazy=True)
```

[ ]:
```
#s_dpc
```

[ ]:
```
#s_dpc.save("datasett/stem_dpc_data.zspy", chunks=(64, 64, 64, 64))
#s_dpc.save("datasett/test_merlin.zspy", chunks=(64, 64, 64, 64))
```

2.1 Åpne dataset

Dette gjøres via `hs.load`, som kan åpne en rekke dataformater, spesielt innenfor elektronmikroskopi. Velg et av STEM-DPC datasettene deres, disse skal ha:

- `.zspy` "fileformat", denne er egentlig en mappe, men skriv den inn som om det er en fil.
- Ha filnavn som inneholder noe med: `stem_dpc`, `STEMDPC`, `LowMag`, `Low_Mag`, `lowmag`, `obj_off` eller `OBJOFF`

Siden disse er ganske store, så husk å bruk `lazy=True`. Lag et objekt som heter `s`.

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

(Det er mulig at dere får en `WARNING` melding om `pyOpenCl`. Dette er en `WARNING` ikke en `ERROR`, så her kan den ignoreres.)

[ ]:
```
s = hs.load('datasett/test_merlin.zspy',lazy=True)
```

Skriv `print(s)` i cellen under, og kjør cellen.

[ ]:
```
print(s)
```

```
<LazyElectronDiffraction2D, title: , dimensions: (257, 256|256, 64)>
```

Her ser vi at dette er et `LazyElectronDiffraction2D` signal. `Lazy` betyr at dataene er ikke overført til RAM, ergo at dataene ennå bare er på harddisken.

Den siste delen, `(256, 256|256, 256)` er dimensjonene til datasettet. Tallene til venstre for `|` er navigasjons-dimensjonene (probe-posisjoner), mens de til høyre er signal-dimensjonene (detektorposisjoner). Ergo, vi har et 4-dimensjonalt datasett, som består av `256 x 256` probe-posisjoner, og `256 x 256` detektorposisjoner.

For å få mer informasjon om dette, kjør `s` i cellen:

[ ]:
```
s
```

[ ]: <LazyElectronDiffraction2D, title: , dimensions: (257, 256|256, 64)>

[ ]:
```
axes_manager = s.axes_manager
print(axes_manager)  # Check axes dimensions and order
```

```
<Axes manager, axes: (257, 256|256, 64)>
            Name |   size |  index |  offset |   scale |  units
================ | ====== | ====== | ======= | ======= | ======
                 |    257 |      0 |       0 |       1 |
                 |    256 |      0 |       0 |       1 |
---------------- | ------ | ------ | ------- | ------- | ------
                 |    256 |      0 |       0 |       1 |
                 |     64 |      0 |       0 |       1 |
```

[ ]:
```
s_T = s.T
s_T.calibrate()  # Prompts you to draw a line
```

```
[########################################] | 100% Completed | 316.11 ms
```

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

[ ]:
```
s = s_T.T
axes_manager = s.axes_manager
print(axes_manager)  # Check axes dimensions and order
```

```
<Axes manager, axes: (257, 256|256, 64)>
            Name |   size |  index |  offset |   scale |  units
================ | ====== | ====== | ======= | ======= | ======
                 |    257 |      0 |       0 |   0.051 |     um
                 |    256 |      0 |       0 |   0.051 |     um
---------------- | ------ | ------ | ------- | ------- | ------
                 |    256 |      0 |       0 |       1 |
                 |     64 |      0 |       0 |       1 |
```

Her ser vi at den totale størrelsen på datasettet er 8 GiB, og at dataene er lagret i 16 usigned integer. Dette gir 2 bytes per datapunkt (8 bits i en byte).

En del av dere har nok en datamaskin som kan takle dette, men la oss prøve å redusere datamengden litt.

VIKTIG: det er veldig lett å kræsje datamaskinen når man holder på med såpass store datasett. Så pass på at dere har lagret ting dere har åpent.

2.2 Plotting av dataen

`s` er et signal objekt som inneholder mange funksjoner. En av disse er `plot`, som lar oss visualisere dataene. Kjør denne funksjonen.

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

```
[########################################] | 100% Completed | 870.76 ms
```

Siden dette er et `lazy` signal, så må HyperSpy kalkulere et navigasjonsbilde ved å hente ut deler (`chunks`) av gangen.

Denne navigeringen kan "hakke" litt, dette fordi alt må leses fra harddisken. Planen nå er å redusere datastørrelsen, slik at vi kan laste alt inn i minnet, men først vil vi utforske datasettet litt for å se hvor mye vi kan redusere datasettet.

Dere får opp to bilder: "navigeringsplot" og "signalplot".

- Tips 1: navigatoren kan gjøres større ved å trykke på `+` knappen på **tastaturet**. Og mindre med å trykke på `-` knappen på **tastaturet**. Dette summerer IKKE flere piksler, men er bare en måte å lettere treffe navigator-markøren.
- Tips 2: dere kan også flytte rundt med pil-tastene.

(Siden folk har litt forskjellige datasett, så er det sannsynlig at ikke alt dette er relevant for alle.) Som dere kanskje har sett allerede, så er bildet i eksemplet over, annerledes enn bildet i plottene deres.

Dette er fordi HyperSpy bare bruker den midterste "chunken" i signal-dimensjonen for å lage navigasjonsbildet. For de fleste datasett, så virker dette greit nok. Men her er det litt suboptimalt, fordi elektronproben flytter seg ganske mye. Så la oss lage et bedre "navigasjonsbilde". For å gjøre dette, så lager vi først et "bright field" bilde av datasettet. Ergo, vi summerer hele diffraksjonsmønsteret i hver eneste probeposisjon. For å gjøre dette bruker vi `sum` funksjonen som er i `s`. Men vi må spesifisere at vi vil summere over de to siste dimensjonene, for å gjøre dette. Bruk `axis=(-1, -2)` i `sum` funksjonen. Resultatet i denne skal så lagres i en ny variabel: `s_sum`

[ ]:
```
s_sum = s.sum(axis=(-1,-2))
```

Så kan vi se hva dette nye signalet er, kjør `print(s_sum)`

[ ]:
```
print(s_sum)
```

```
<LazySignal, title: , dimensions: (257, 256|)>
```

Her ser vi at `s_sum` har 2 navigasjonsdimensjoner, og 0 signaldimensjoner. Disse vil vi "flippe", ergo å få 0 navigasjonsdimensjoner, og 2 signaldimensjoner.

Gjør dette ved å bruk `transpose` funksjonen i `s_sum`, og lagre den til en ny variabel `s_nav`. Så kjør `print(s_nav)` for å se hvordan den ser ut.

[ ]:
```
s_nav = s_sum.T
print(s_nav)
```

```
<LazyElectronDiffraction2D, title: , dimensions: (|257, 256)>
```

Nå dar den riktige dimensjoner, men den er ennå `LazyElectronDiffraction2D`. Ergo, vi har ikke egentlig gjort alle operasjonene ennå. For å gjøre dette, kjør `compute` funksjonen i `s_nav`. Så kjør `print(s_nav)`

[ ]:
```
s_nav.compute()
print(s_nav)
```

```
[########################################] | 100% Completed | 1.19 sms
<ElectronDiffraction2D, title: , dimensions: (|257, 256)>
```

Nå er `ElectronDiffraction2D`, ergo at vi har gjort alle kalkuleringene, og lastet dataene inn i minnet.

Kjør så `plot` funksjonen i `s_nav`.

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

Dette er et "bright field" bilde av strukturen, som vi skal bruke som navigasjonsbilde.

Dette gjøres ved å sette `navigator` attribute i `s` lik `s_nav`.

[ ]:
```
s.navigator==s_nav
```

[ ]:
```
<ElectronDiffraction2D, title: , dimensions: (|257, 256)>
```

Deretter kan vi bruke `plot` funksjonen i `s`. Da blir `s_nav` automatisk brukt som navigasjonsbilde.

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

Siden vi bare er interessert i senter-disken, så kan vi fjerne alt "tomrommet" (det grønne i bildet her) hvor elektronstrålen ikke er. Ergo: vi beskjærer datasettet, slik at vi bare får de delene vi bryr oss om.

Et vanlig problem, er at elektronstrålen flytter seg som funksjon av probe-posisjon. Så vi kan ikke bare beskjære akkurat rundt der den er i en enkeltposisjon, vi må ha litt "ekstra" rom på sidene.

- Plasser navigatoren midt i datasettet, som vist i bildet over.
- Se hva `x` og `y` er i senterpunktet av disken (øverst til høyre i signal plottet)
- Bruk `isig` til å beskjære. Syntaksen er: `s.isig[x0:x1, y0:y1]`, hvor dere kan for eksempel ha +- 50 rundt senterposisjonen. Ergo at det beskjærte området blir 100 x 100 piksler tilsammen.
  - Eksempel: `s.isig[128 - 50 : 128 + 50, 128 - 50 : 128 + 50]`
- Lagre dette som en ny variabel: `s1`.

Hvis deler av datasettet er "dekket" av tykke deler av prøven, som dere ikke bryr dere om, så bare gjør dette med de områdene som er tynne nok. Hvis du har sånne "uinteressante" områder, så bruk `inav` til å fjerne dem på en tilsvarende måte. Skriv `s1` i cellen under, og kjør cellen.

[ ]:
```
s1 = s.isig[90:150, 0:62]
s1
```

[ ]:
```
<LazyElectronDiffraction2D, title: , dimensions: (257, 256|60, 62)>
```

Her ser vi at dette er et `LazyElectronDiffraction2D` signal, men med færre detektor-posisjoner! Hvis dere brukt 50 piksler som eksemplet ovenfor, så vil dette nye signalet `s1` bare være 15% av `s` sin størrelse.

Her kan vi gjenbruke navigasjonsbildet fra tidligere, via `navigator` attributen. For å sjekke hvordan dette ser ut, så bruk: `s1.plot()`

[ ]:
```
s1.navigator
s1.plot()
```

```
[########################################] | 100% Completed | 429.32 ms
```

Sjekk at den hele senterdisken ennå er i datasettet, ved å flytte navigatoren til de ytre hjørnene.

Nå er navigeringen mye raskere, fordi vi laster en mye mindre del av datasettet inn i minnet per probeposisjon.

Hvis dette ser bra ut så bruk `compute` funksjonen i `s1`.

VIKTIG: dette vil laste hele `s1` datasettet inn i minnet, og hvis du ikke har gjort de forrige stegene riktig, så kan det kræsje datamaskinen din!

Nå vil plottingen være mye raskere, siden alt er i minnet. Bruk `plot` i `s1`, for å se hvordan prøven og datasettet ser ut.

[ ]:
```
s1.compute()
s1.plot()
```

```
[########################################] | 100% Completed | 967.51 ms
```

2.3 Magnetisk kontrast

Nå som datasettet er litt mer håndterbart, så kan vi prøve å se de magnetiske domenene.

En enkel måte å gjøre dette på, er å bytte om på "navigasjon" og "signal" dimensjonene. Ergo: istedet for at vi endrer på probe-posisjonen, så endrer vi heller på detektorposisjonen.

Kjør: `s1.T.plot()`, og flytt navigatoren rundt senterstrålen. Dette vil se litt rart ut, på grunn av at elektronstrålen flytter på seg, men dere burde kunne se litt magnetisk kontrast på grensen mellom de lyse og mørke områdene.

[ ]:
```
s1.T.plot()
```

2.4 Mer avansert analyse

En litt mer avansert måte å analysere dette, er å bruk `get_direct_beam_position` funksjonen i `s1`, med parameter `center_of_mass`. Lagre dette som `s1_bs`. Dette regner ut hvor senter-posisjonen er for alle probe-posisjonene. Dette gir et `BeamShift` signal, hvor x- og y-posisjonene er i navigasjonsposisjonene. Bruk `plot` i `s1_bs`.

[ ]:
```
s1_bs = s1.get_direct_beam_position(method = 'center_of_mass')
s1_bs.plot()
```

[ ]:
```
[<Axes: title={'center': 'x-shift'}, xlabel=' axis (um)', ylabel=' axis (um)'>,
 <Axes: title={'center': 'y-shift'}, xlabel=' axis (um)', ylabel=' axis (um)'>]
```

Her ser vi at vi har et problem med at senter-strålen har flyttet seg, som gir et "plan" i både x- og y-retningen.

Dette kan korrigeres ved å bruk `get_linear_plane` funksjonen i `s1_bs`.

Bruk denne, og lagre denne som en nytt signal: `s1_bs_lp`.

[ ]:
```
s1_bs_lp = s1_bs.get_linear_plane()
```

Så lag et nytt signal `s1_bs_corr` ved å ta `s1_bs` minus `s1_bs_lp`. Så plot `s1_bs_corr`, ved å bruke `get_magnitude_phase_signal().plot()`.

[ ]:
```
s1_bs_corr = s1_bs - s1_bs_lp
s1_bs_corr.get_magnitude_phase_signal().plot()
```

Dette skal se noe ut som dette:

3 Andre visualiseringer

Styrke på magnetismen: `s1_bs_corr` sin funksjon `get_magnitude_signal`

[ ]:
```
s1_bs_corr.get_magnitude_signal()
```

```
[########################################] | 100% Completed | 260.61 ms
```

[ ]:
```
<Signal2D, title: Magnitude of , dimensions: (|257, 256)>
```

Histogram av magnetiske vektorer, `get_bivariate_histogram`

[ ]:
```
s1_bs_corr.get_bivariate_histogram()
```

[ ]:
```
<Signal2D, title: Bivariate histogram of , dimensions: (|200, 200)>
```

[ ]:
```
s1_bs_corr.axes_manager
```

[ ]:
```
<Axes manager, axes: (257, 256|2)>
            Name |   size |  index |  offset |   scale |  units
================ | ====== | ====== | ======= | ======= | ======
                 |    257 |      0 |       0 |   0.051 |     um
                 |    256 |      0 |       0 |   0.051 |     um
---------------- | ------ | ------ | ------- | ------- | ------
   Beam position |      2 |      0 |      90 |       1 |
```

[ ]:
```
s1_bs_corr.axes_manager.gui()
```

```
HBox(children=(Accordion(children=(VBox(children=(HBox(children=(Label(value='Name'),␣
↪Text(value='')), layout=…
```

[ ]:
```
#s1_bs_corr.T.calibrate()  # Prompts you to draw a line
```

3.1 Plotting av disse dataene

Nå kan dette kombineres med kunnskapen og koden dere brukte i TEM-bildedata notebooken, og FIB notebooken, til å lage en figur som ligner på den i starten av denne Jupyter Notebooken.

For å få disse fargeplottene i en matplotlib-figur, så først lag en `fig` og en `ax` via matplotlib.

Første importer `pyxem` som `pxm`.

Så bruk `pxm.utils.plotting.plot_beam_shift_color` med `s1_bs_corr` og `ax=ax`.

Tips: posisjonen til fargehjulet kan styres med `ax_indicator` parameteren, se docstringen for mer informasjon.

Husk å legge til en "scalebar", på samme måte som i tidligere dataøvinger.

[ ]:
```
%matplotlib widget
import pyxem as pxm
import matplotlib.pyplot as plt
# Create a figure and axes for plotting
fig, ax = plt.subplots(figsize=(5,5))

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': 3, 'label': '3 um', 'loc': 4, 'frameon': False,␣
 ↪'color': 'white', 'size_vertical': 0.4, '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)
ax.annotate(text='A', xy=(0.03, 0.93), xycoords='axes fraction',fontsize=20,␣
 ↪color='white', fontweight='bold')

ax_indicator = fig.add_axes([0.13, 0.01, 0.15, 0.355])  # Adjust [left, bottom,␣
 ↪width, height] as needed
# Assuming s1_bs_corr contains the required data
pxm.utils.plotting.plot_beam_shift_color(s1_bs_corr, ax=ax,␣
 ↪ax_indicator=ax_indicator)
fig.savefig('05_STEM.png',dpi=600,bbox_inches='tight')##,bbox_inches='tight',␣
 ↪pad_inches=0
plt.close(fig)
```