usage_example

July 1, 2021

1 mreels.py usage and examples

[1]:
```
#import the module the standard python way
import mreels

#importing other packages for examples
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

1.1 Dependencies

The package needs a few other python modules to be available in the current environment, namely: - Numpy, for array calculations - Scipy, for Fourier transforms and 2-dimensional convolution - Matplotlib, for plotting - Ncempy, for importing Gatan's proprietary .dm4 files - Tqdm, for progressbars so you know that the code is just slow, not broken ;) .

1.2 Starting out

1.2.1 The data object

To start using the the mreels package you first need to create a data python object from the .dm4 file. The data object will be used by most functions since it holds all data and necessary metadata of the EFTEM stack.

To create a data object you need the .dm4 file in the same directory as your python file. Then you can call:

```
data = mreels.MomentumResolvedDataStack('string', preferred_frame)
```

with: - string : str , name of the dm4 file - preferred_frame : int , index of the energy slice which will be used as reference point for other functions

preferred does not need to be set right away, but is used by build_axes so setting pref_frame to a energy slice with clear diffraction spots and bright centre gives better results.

to change to preferred frame you can call:

```
data.pref_frame = 25 #example
```

After the data-object is initialised a few functions can be called which change or append data to the data-object:

- `data.remove_neg_val()`, to raise all intensities by the minimum amount to make all intensities positive
- `data.rem_neg_el()`, removes the negative energy-loss slices from the EFTEM stack
- `data.build_axes()`, builds all the axis corresponding to the shape of `data.stack.shape`

[2]:
```
data = mreels.MomentumResolvedDataStack('inse_example_stack.dm4', pref_frame=10)

shape = data.stack.shape
print(shape)

data.rem_neg_el() #removes slices corresponding to negative energy-losses

shape = data.stack.shape
print(shape)

data.build_axes()
print("Energy losses are: \n", data.axis0)
```

```
(40, 2048, 2048)
(37, 2048, 2048)
Energy losses are:
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35.
 36.]
```

1.2.2 Plotting energy-slices

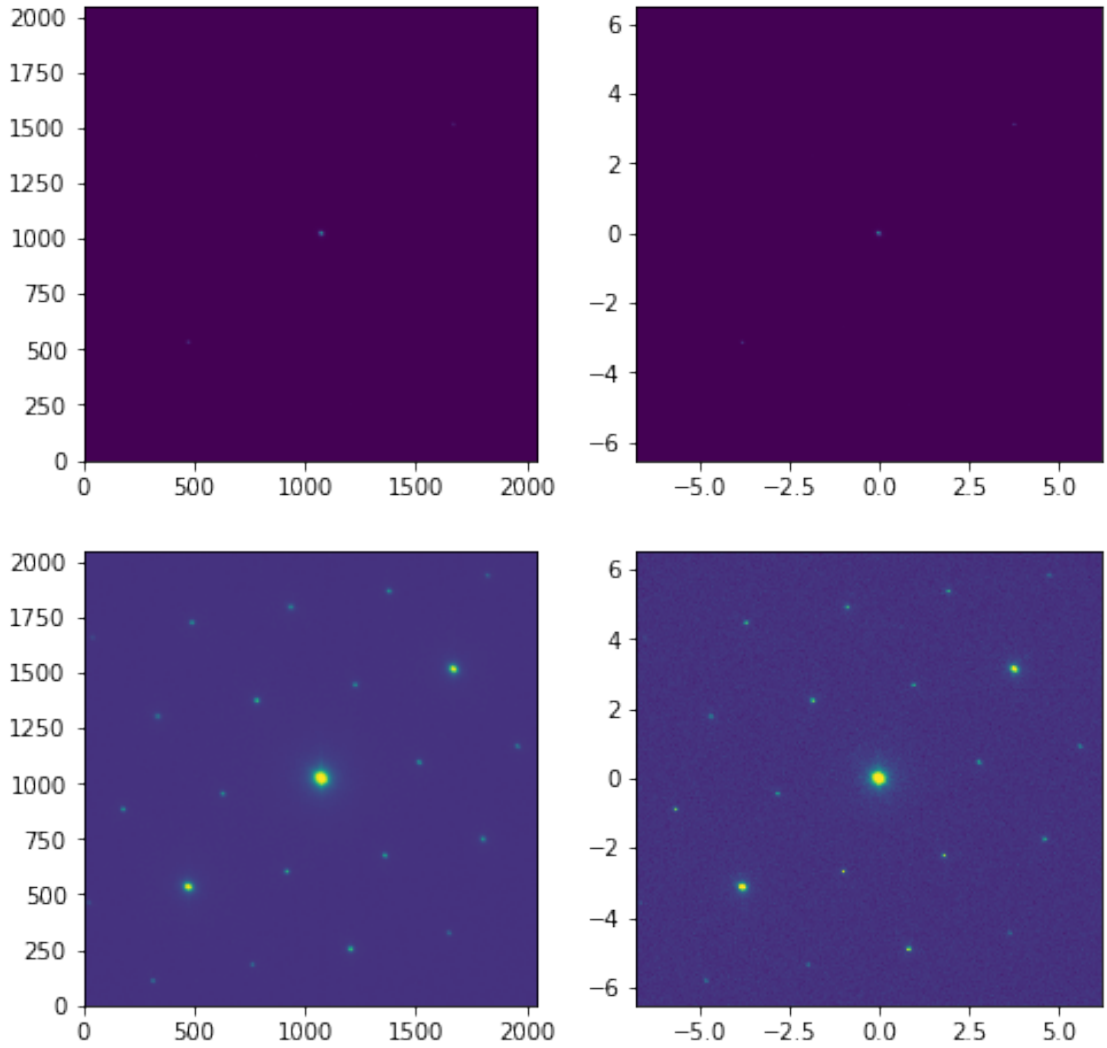
The EFTEM stack can be called from the data object by calling `python data.stack`, this returns a numpy.ndarray object with 3-axis. Axis-0 is the energy-loss and axis-1 and -2 are the first and second reciprocal space axis.

If the example above was executed you can see that there were 40 energy-loss slices and 2048 momentum coordinates for the first and second reciprocal space axis. The `python data.rem_neg_el()` removed 3 energy-slices that corresponded to negative energy losses.

Plotting can be done using the matplotlib plotting library.

[3]:
```
ax1 = data.axis1
ax2 = data.axis2
stack = data.stack

fig, ax = plt.subplots(2, 2)

ax[0,0].imshow(stack[10,:,:].T, origin='lower')

ax[0,1].pcolorfast(ax1, ax2, stack[10,:,:].T)

sig = mreels.sigmoid
```

```
ax[1,0].imshow(sig(stack[10,:,:].T), origin='lower')

ax[1,1].pcolorfast(ax1, ax2, sig(stack[10,:,:].T))

plt.gcf().set_size_inches(8,8)
plt.show()
```



1.2.3 Finding features

After plotting a 'normalised' stack slice serveral diffraction peaks are visible. To find the centres of these diffraction peaks in their index coordinates a list with their false coordinates needs to be build by guessing the locations.

Coordinates are indexed as the stack was supplied, thus in (energy loss, ax1, ax2) format. For guessing peaks this means denoting the 'x-axis'value first.

A false centres list would be:

`[(500, 500), (900, 600)]` for the lower left bright peak and the dimmer peak to the right respectively.

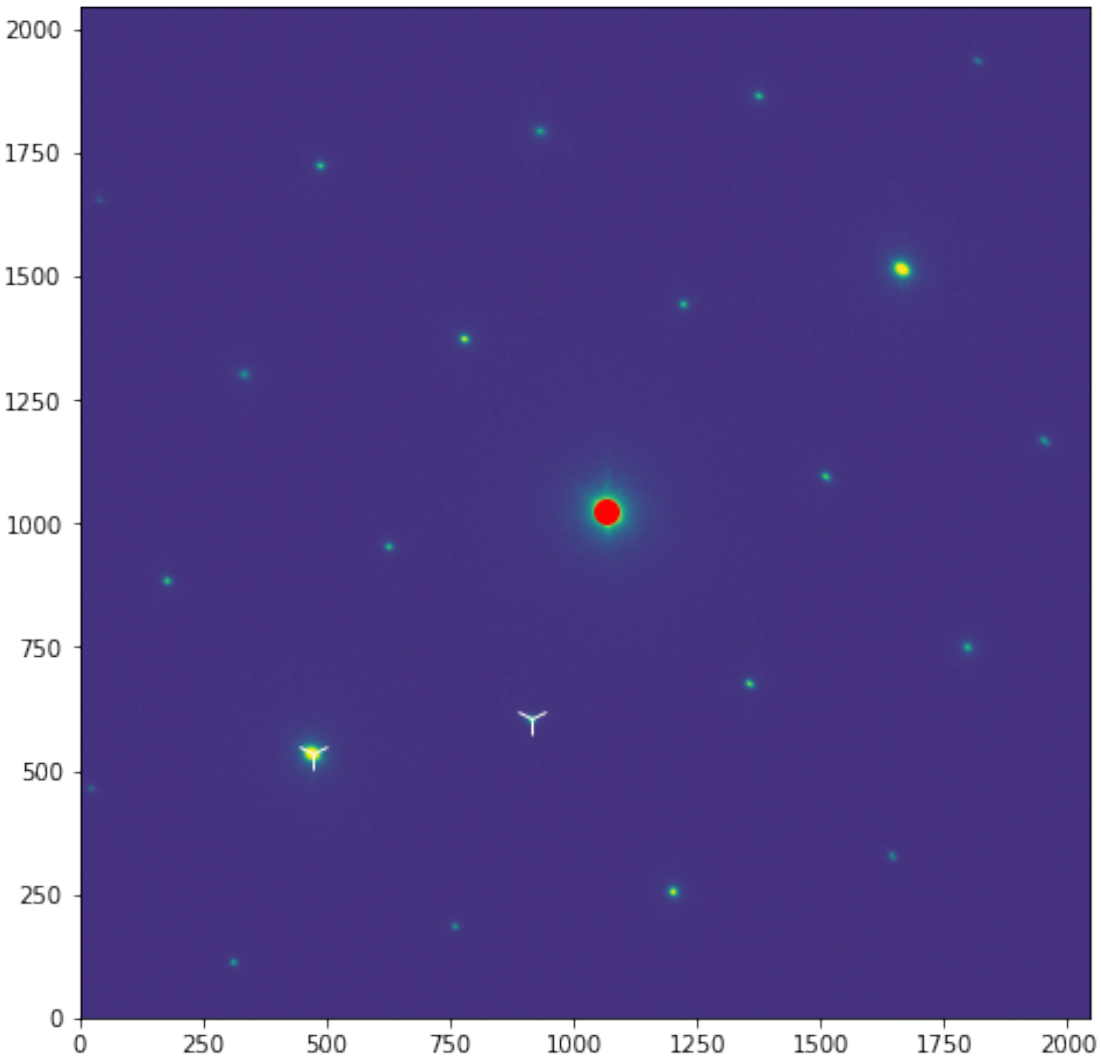
Finding the actual centres would then be as easy as specifying a energy-loss slice in which to find peaks, passing the list and specifying `leeway=` which is the search block size.

The centre of the diffraction image is assumed to be the brightest pixel in a slice, its coordinates can be requested by calling: `data.get_centre()`

[4]:
```
fls_centres = [(500, 500), (900, 600)]

tr_centres = mreels.get_true_centres(stack[10], fls_centres, leeway=100)

print("True centres: \n", tr_centres)

centre = data.get_centre()

print("Stack centre: \n", centre)
```

```
True centres:
 [471 535 914 604]
Stack centre:
 (1068, 1025)
```

[5]:
```
fig, ax  =plt.subplots(1,1)

ax.imshow(sig(stack[10].T), origin='lower')
ax.plot( 471, 535, marker='1', color='white', markersize=14)
ax.plot( 914, 604, marker='1', color='white', markersize=14)
ax.plot( 1068, 1025, marker='o', color='red', markersize=10)

plt.gcf().set_size_inches(8,8)
plt.show()
```



1.2.4 Aligning the stack

The EFTEM stack in the data object can be aligned by calling `data.correct_drift()`. This function corrects for 2D-drift of the diffraction pattern, thus not for: warp, rotation, skewing etc.

It does this by means of Fourier phase correlation on one enhanced slice at a time, looping over the stack and using `data.pref_frame` as a reference slice.

[6]:
```
data.correct_drift()
```

```
Aligning EFTEM stack: 100%|      | 37/37 [00:17<00:00,  2.11it/s]
```

1.2.5 Extracting information

Information that can currently be extracted from the data object is limited to: - q-EELS spectra, not automated but easily done by slicing. - q-EELS maps using one of three functions.

- Radial averaging This method allows for the averaging of q-EELS spectra of a certain momentum transfer radially outwards from the centre.

Usage:

```
qmap, qaxis = mreels.get_qeels_data(data, r1, ringsize, pref_frame, forward_peak,
                method='radial', threads, starting_point, peak_width
                )
```

with:

- `data`, the data object
- `r1`, the staring radius in pixel lengths such that if the separation from the centre in x and y = 10 that r = sqrt(10**2** + **10**2)
- `pref_frame`, frame that will be used to find the peaks and centre
- `forward_peak`, not used if `method='radial'`
- `method='radia'`, to use radial averaging
- `threads`, number of cpu threads to use. function distributes energy-loss slices to threads so it should not be RAM limited.
- `starting_point`, will be used as centre. Can be left `None` to use stack centre
- `peak_width`, unused in `method='radial'`
- Line averaging This method allows for the averaging of q-EELS spectra of certain momentum transfer angle limited radially outwards from the centre in the direction of a specified point.

Usage:

```
qmap, qaxis = mreels.get_qeels_data(data, r1, ringsize, pref_frame, forward_peak,
                method='line', threads, starting_point, peak_width
                )
```

with:

- `data`, the data object
- `r1`, the staring radius in pixel lengths such that if the separation from the centre in x and y = 10 that r = sqrt(10**2** + **10**2)
- `pref_frame`, frame that will be used to find the peaks and centre
- `forward_peak`, peak to which will be integrated
- `method='line'`, to use line averaging
- `threads`, number of cpu threads to use. function distributes energy-loss slices to threads so it should not be RAM limited.
- `starting_point`, will be used as centre. Can be left `None` to use stack centre
- `peak_width`, width of the `forward_peak` in pixels. Needs to be set only if `forward_peak` is not a bright well-defined peak. Otherwise it will be determined as the amount of pixels from the `peak_centre` by which 1/3rd of the peak intensity is reached

- Line slicing This methods allows the user to extract all q-EELS spectra and corresponding momenta transfers in a line from point to point.

Usage:

```
qmap, qaxis = mreels.get_qeels_slice(data, point, use_k_axis, starting_point)
```

with:

- `data`, the data object
- `point`, the forward point to slice towards.
- `use_k_axis`, ignore this for know
- `starting_point`, to point from which to slice

1.2.6 Plotting a q-EELS map

A q-EELS map can be simply plotted using the function:

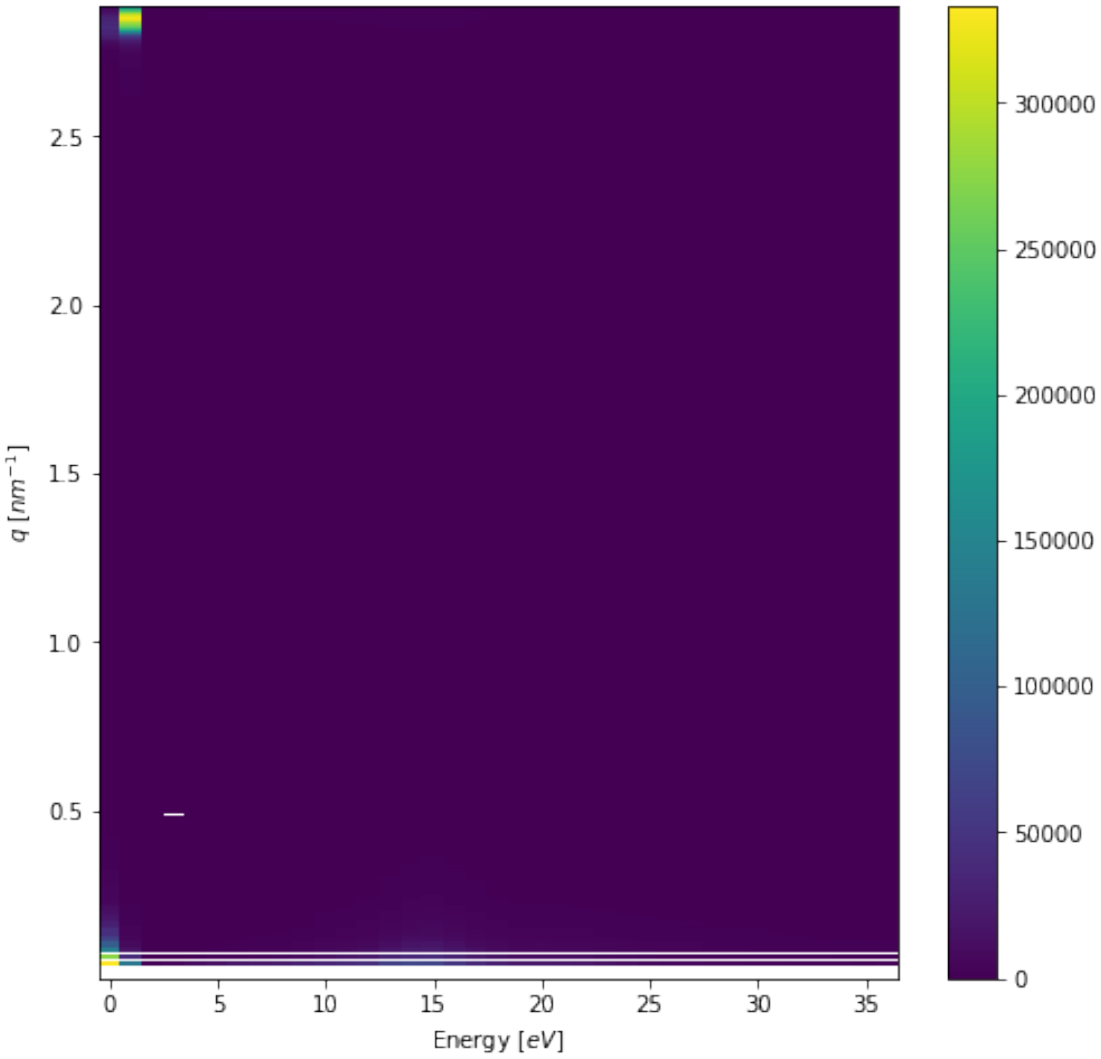
```
mreels.plot_qeels_data(data, qmap, qaxis, prefix, save)
```

with: - `data`, data object - `qmap`, a qmap supplied by one of the previous functions - `qaxis`, qaxis supplied by previous functions corresponding to qmap - `prefix`, string to prefix to saved file if `save=True` also functions as plot title - `save`, boolean to toggle saving

[7]:
```
fwd_peak0 = (914, 604)
fwd_peak1 = (471, 535)

qmapline, qaxisline = mreels.get_qeels_data(data, 100, 1, 10, fwd_peak0,␣
↪method='line', threads=12)
```

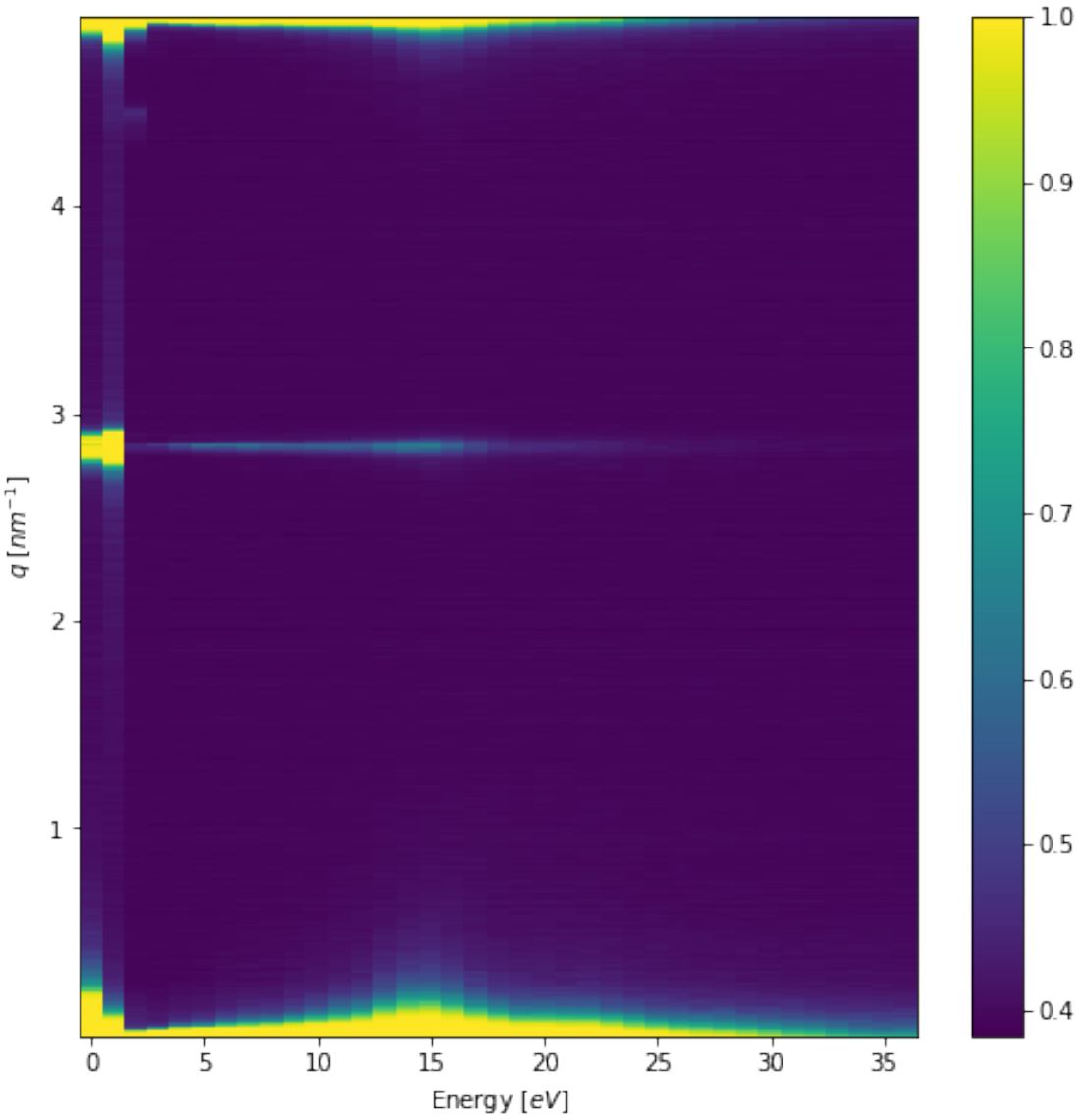
```
Momentum axis: 100%|      | 454/454 [00:07<00:00, 64.73it/s]
  0%|          | 0/37 [00:00<?, ?it/s]c:\Users\Jeroe\Documents\Python\CBL-ML\MR-
EELS\mreels.py:327: RuntimeWarning: invalid value encountered in double_scalars
  integral = np.sum(integration_area)/np.sum(entries)
c:\Users\Jeroe\Documents\Python\CBL-ML\MR-EELS\mreels.py:327: RuntimeWarning:
divide by zero encountered in double_scalars
  integral = np.sum(integration_area)/np.sum(entries)
100%|     | 37/37 [00:56<00:00,  1.52s/it]
```

[8]:
```
mreels.plot_qeels_data(data, qmapline, qaxisline, '')
```

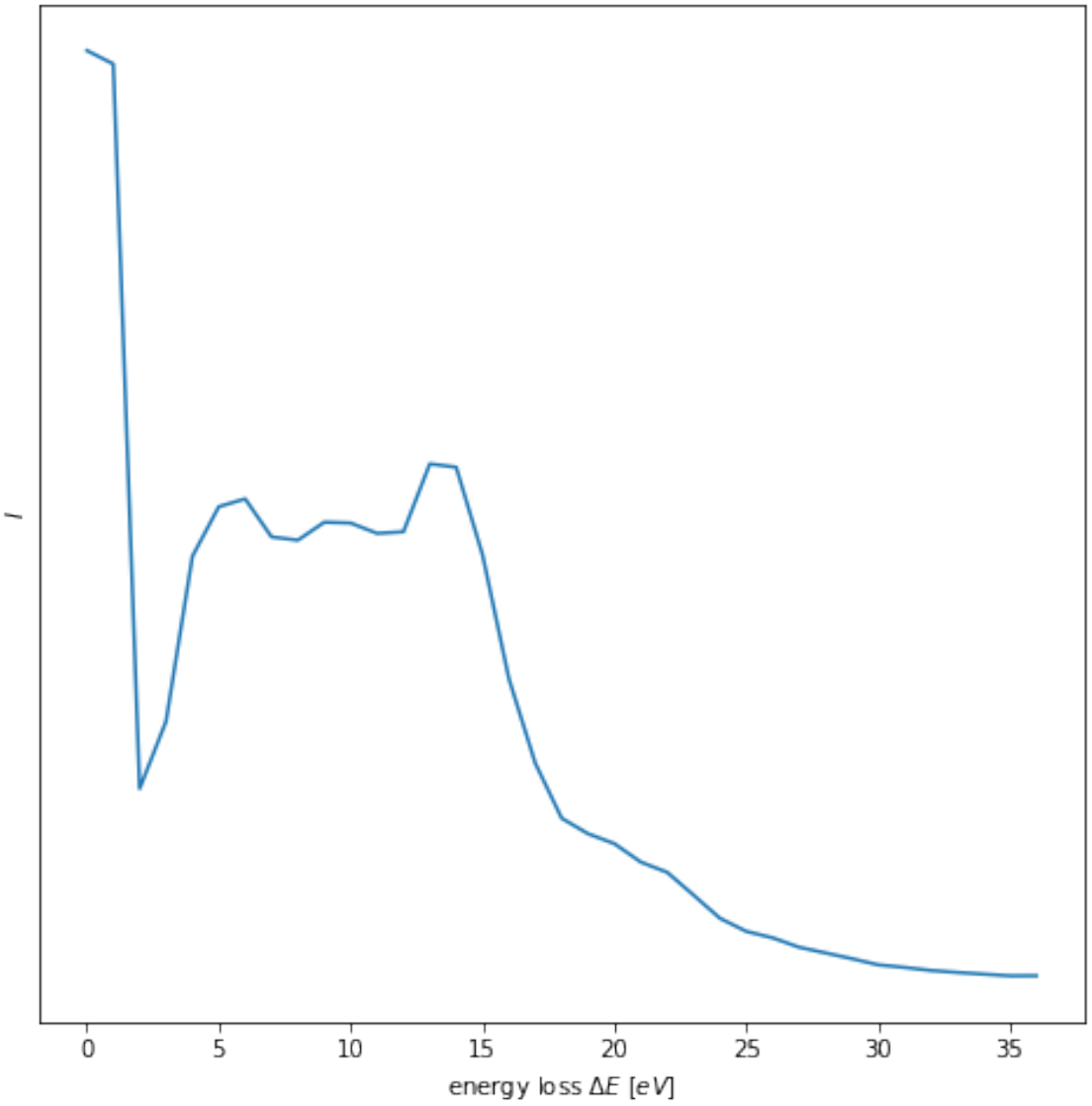


[9]:
```
# q-EELS slice maps can be easily chained

qmapslice0, qaxisslice0 = mreels.get_qeels_slice(data, fwd_peak0)
qmapslice1, qaxisslice1 = mreels.get_qeels_slice(data, fwd_peak1,␣
↪starting_point=fwd_peak0)

qmapslice = np.append(qmapslice0, qmapslice1[1:], axis=0)
qaxisslice = np.append(qaxisslice0, qaxisslice1[1:])
```

[10]:
```
mreels.plot_qeels_data(data, sig(qmapslice, (20,50)), qaxisslice, '')
```



[11]:
```
eax = data.axis0

fig, ax = plt.subplots(1,1)
ax.plot(eax, stack[:,centre[0], centre[1]])
ax.set_xlabel(r"energy loss $\Delta E$ [$eV$]")
ax.set_ylabel(r"$I$")
ax.set_yticks([])

plt.gcf().set_size_inches(8,8)
plt.show()
```



1.2.7 Batson correction

Batson correction uses an image spectrum of the same sample to remove the zero-loss peak and to try and remove plural scattering from the individual q-EELS spectra in the q-EELS map.

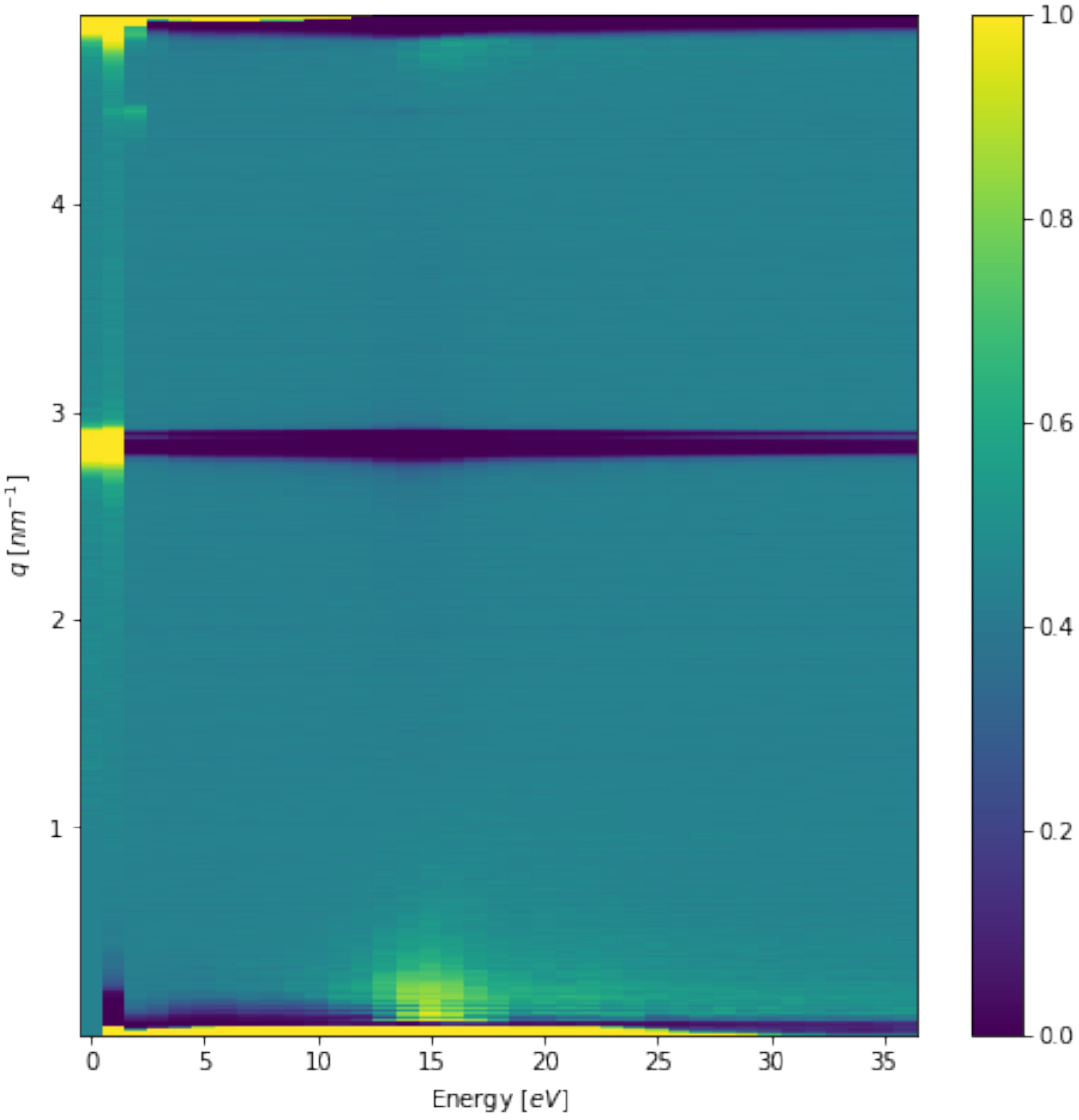
Usage:

```
batmap = mreels.batson_correct(data, energy_window, qmap, imspec)
```

with: - `data`, the data object - `energy_window`, the size in eV of the zero-loss peak - `qmap`, the to be corrected q-EELS map - `imspec`, the image spectrum

[12]:

```
imspec = np.sum(stack[:,centre[0]-100:centre[0]+100,centre[1]-100:
↪centre[1]+100], axis=(2,1)) #simulating an image spectrum, use an actual␣
↪image spectrum if you have one

batmap = mreels.batson_correct(data, 3, qmapslice, imspec)

mreels.plot_qeels_data(data, sig(batmap, (20,50)), qaxisslice, '')
```



1.2.8 Peak tracking

Peaks in the spectra can be tracked accross momentum space using the following functions:

`pqmap, pqaxis = mreels.pool_qmap( qmap, qaxis, poolsize)`

with: - `qmap`, a q-EELS map - `qaxis`, a q-EELS momentum axis - `poolsize`, the size of pooling `poolsize/2` spectra before and after are pooled together.

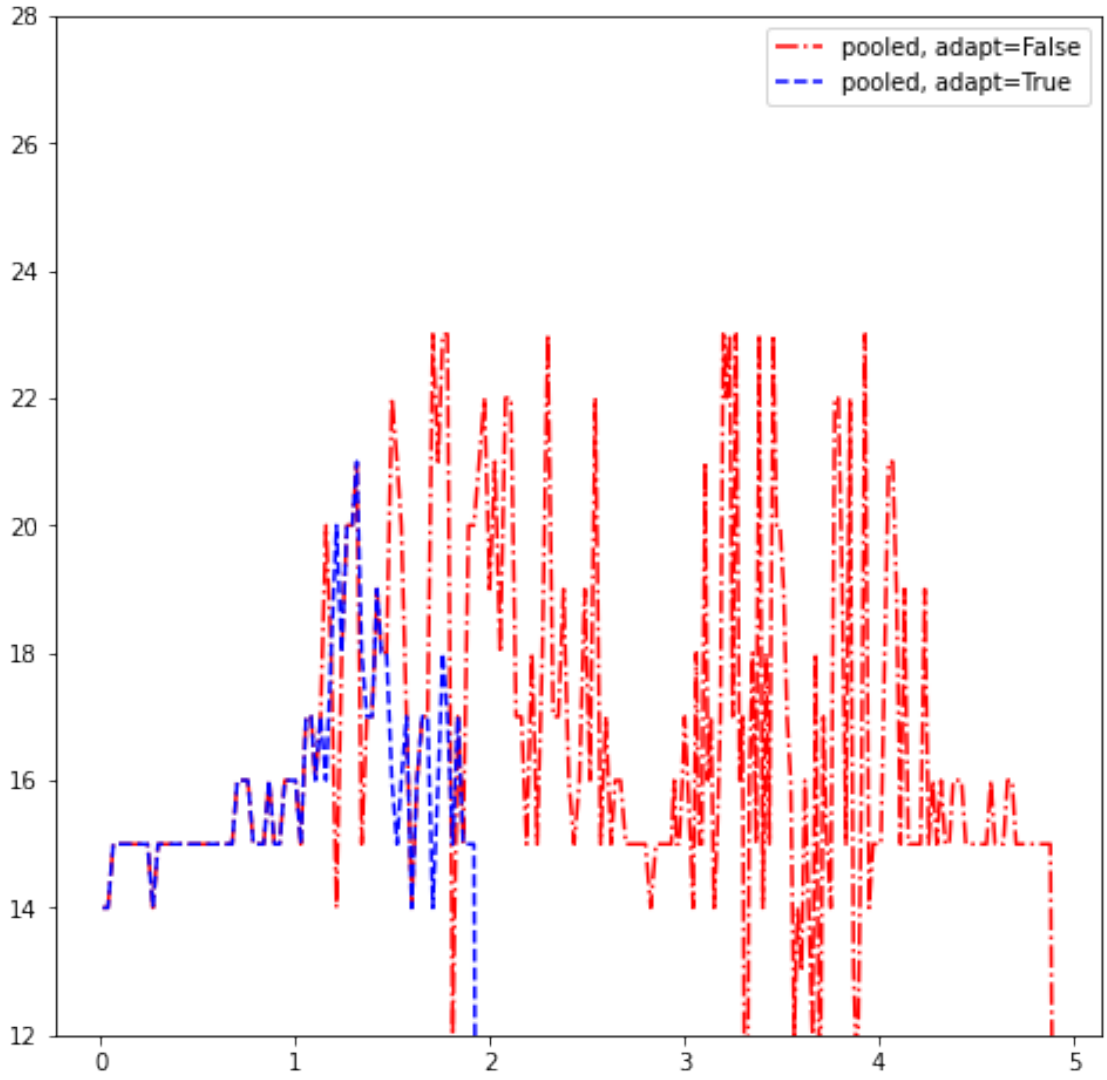
`ppos, perr = mreels.find_peak_in_range(qmap, centre, windows_size, adaptive_range)`

with; - `qmap`, a (pooled) q-EELS map - `centre`, index around which to find the Peak - `window_size`, size of window in which to search for peak centred around centre - `adaptive_range`, boolean toggle for adaptive range. If `True` the centre for next search in a q-EELS spectra is the peak location in this spectra

Note: perr is currently poorly defined

[13]:
```
pqmap, pqaxis = mreels.pool_qmap(qmapslice, qaxisslice, 4)

ppos, perr = mreels.find_peak_in_range(pqmap, np.argwhere(eax==14)[0][0], 20)
pposadapt, perradapt = mreels.find_peak_in_range(pqmap, np.
 ↪argwhere(eax==14)[0][0], 6, adaptive_range=True)


fig, ax = plt.subplots(1,1)
ax.plot(pqaxis, eax[ppos], label=r'pooled, adapt=False', linestyle='-.',␣
 ↪color='red')
ax.plot(pqaxis, eax[pposadapt], label=r'pooled, adapt=True', linestyle='--',␣
 ↪color='blue')
plt.legend()

plt.ylim([12,28])

plt.gcf().set_size_inches(8,8)
plt.show()
```

```
c:\Users\Jeroe\Documents\Python\CBL-ML\MR-EELS\mreels.py:815: RuntimeWarning:
divide by zero encountered in double_scalars
  perr = np.append(perr, 1/(tmp[0][0]-np.mean(search_slice)))
```


pooled, adapt=False
pooled, adapt=True
28
26
24
22
20
18
16
14
12
0
1
2
3
4
5