Spectral_image class

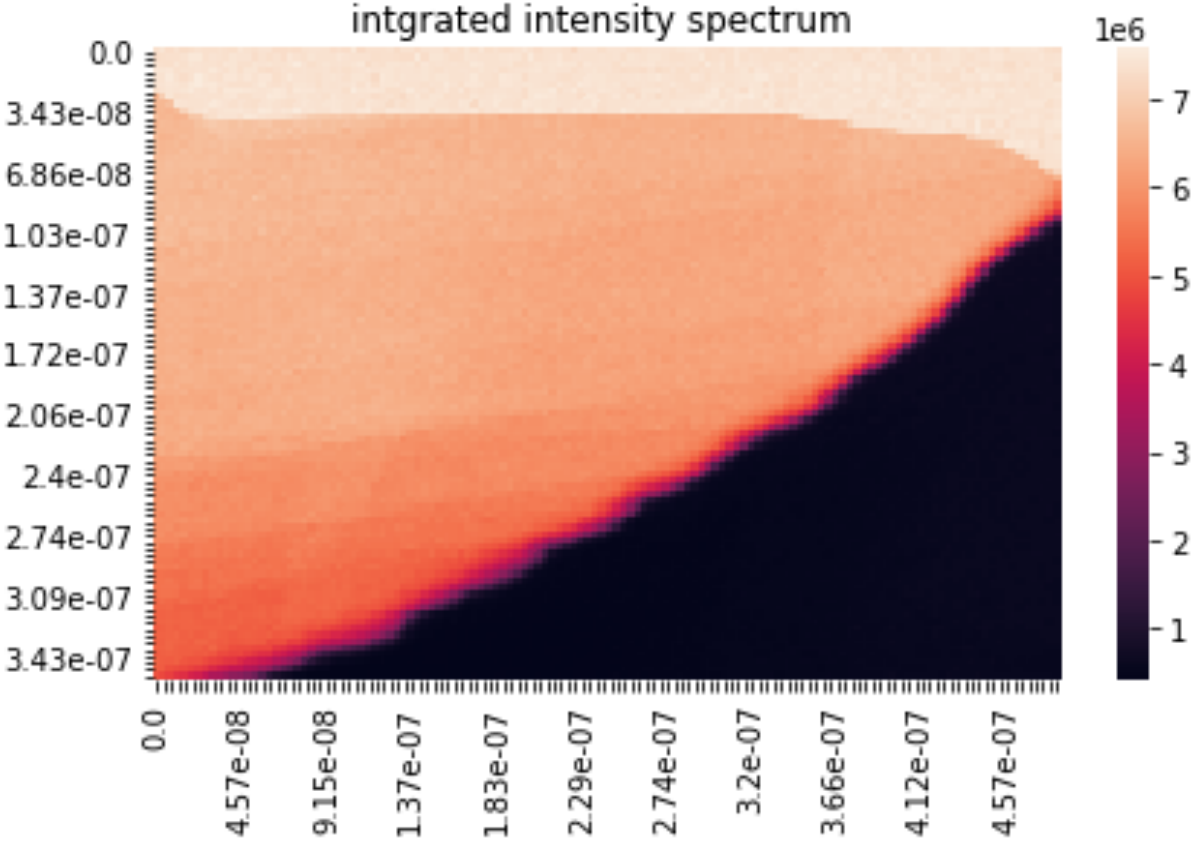
Loading data

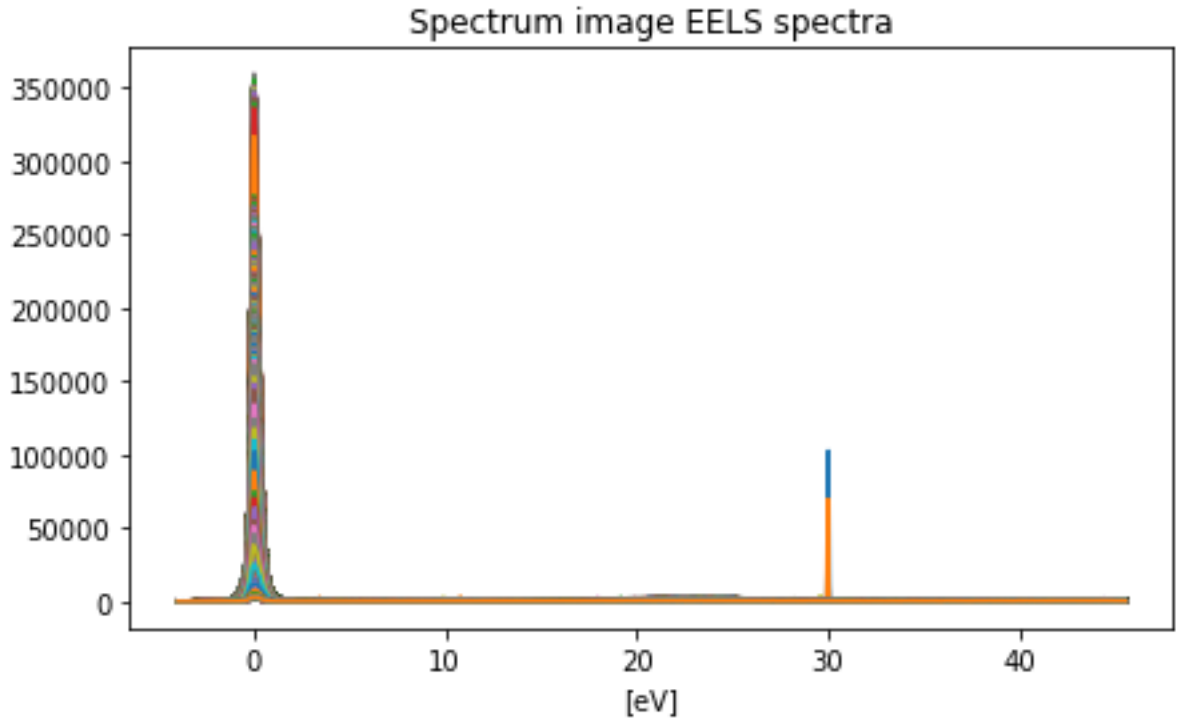
```
>>>im = Spectral_image.load_data(‘path/to/dmfile.dm#’)
>>>im.__dict__.keys()
dict_keys(['data', 'ddeltaE', 'deltaE', 'pixelsize', 'y_axis', 'x_axis’])
>>>dir(im)
['DIELECTRIC_FUNCTION_NAMES’, 'EELS_NAMES','IEELS_NAMES','ZLP_NAMES’, '__class__’,
'__delattr__’, '__dict__’, '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__le__', '__len__', '__lt__', '__module__', '__ne__’, '__new__’, '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'a_0', 'c', 'calc_ZLPs', 'calc_ZLPs_gen2', 'calc_axes',
'calculate_general_ZLPs', 'cluster', 'clustered', 'clusters', 'crossings', 'crossings_im',
'cut', 'cut_image', 'data', 'ddeltaE’, 'deconvolute', 'deltaE', 'determine_deltaE',
'get_data', 'get_deltaE', 'get_metadata', 'get_pixel_signal', 'get_prefix', 'h_bar',
'im_dielectric_function', 'image_shape', 'kramers_kronig_hs', 'l', 'load_data', 'm_0',
'make_model', 'pixelsize', 'plot_all', 'plot_sum', 'samenvoegen', 'shape', 'smooth',
'x_axis', 'y_axis']
```

Loading data

```
>>>im = Spectral_image.load_data(‘path/to/dmfile.dm#’)
>>>im.__dict__.keys()
dict_keys(['data', 'ddeltaE', 'deltaE', 'pixelsize', 'y_axis', 'x_axis’])
>>>dir(im)
['DIELECTRIC_FUNCTION_NAMES’, 'EELS_NAMES','IEELS_NAMES','ZLP_NAMES’, '__class__’,
'__delattr__’, '__dict__’, '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__le__', '__len__', '__lt__', '__module__', '__ne__’, '__new__’, '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'a_0', 'c', 'calc_ZLPs', 'calc_ZLPs_gen2', 'calc_axes',
'calculate_general_ZLPs', 'cluster', 'crossings', 'crossings_im', 'cut',
'cut_image', 'data', 'ddeltaE’, 'deconvolute', 'deltaE', 'determine_deltaE',
'get_data', 'get_deltaE', 'get_metadata', 'get_pixel_signal', 'get_prefix', 'h_bar',
'im_dielectric_function', 'image_shape', 'kramers_kronig_hs', 'l', 'load_data',
'm_0', 'make_model', 'pixelsize', 'plot_all', 'plot_sum', 'samenvoegen', 'shape',
'smooth', 'x_axis', 'y_axis']
```

Look at the image:






```
>>> im.plot_sum()
```

```
>>>im.plot_all()
```

NB: multiple options: including normalisation, plotting not the EELS function, but some other pixel specific (e.g., dielectric function), altering ranges, etc

Some alterations on the data

- Cut image to rectangle ranging from pixel a trough pixel b in width and from pixel c trough pixel d in height
- Changes the data, outside pixels are lost

```
>>> im.cut_image([a,b+1], [c,d+1])
```

Smoothing signal

- Smooths energyspectra, by default with henning window of length 10
- Overwrites data unless `keep_original = True` is given along

```
>>> im.smooth(window_len= 50, window = 'flat')
```

Now we can get calculating!

- Calculate general (i.e. not matched with spectrum) ZLPs:
- NB: this will change when ZLP fitting is incorperated
- Saved as attribute

```
>>>im.calc_gen_ZLP2()
```

Calculate all ZLPs for single pixel

- Matches all the general ZLPs with the spectrum of pixel [i,j]

```
>>>ZLPs_ij = im.calc_ZLPs(i,j)
```

Deconvolute spectrum

- Deconvolutes spectrum of pixel [i,j] with ZLP_k to find the single scattering distribution $S_{i,j,k}(E)$

```
>>>S_E_ijk = im.deconvolute(i,j,ZLPs_ij[k])
```

Dielectric function of single scattering distribution

- Calculates the dielectric function, the thickness of the sample and the surface scattering distribution for single scattering distribution of pixel [i,j] with ZLP_k ($S_{i,j,k}(E)$):

```
>>>df_ijk, t_ijk, SS_E_ijk =
im.kramers_kronig_hs(self, I_EELS, N_ZLP =
np.sum(ZLPs[k]))
```

Caculate the average dielectric functions etc of complete image

- Calculates the average and standard deviation of the dielectric function, the thickness of the sample and the surface scattering distribution for each pixel
- Saved as attributes

```
>>>im.im_dielectric_function()
```

Evalutate crossings

- Evaluates the crossings of the real part of the average dielectric function of each pixel from negative to positive
- Saves crossings energies in `im.crossings_E`, and number of crossings in `im.crossings_n`

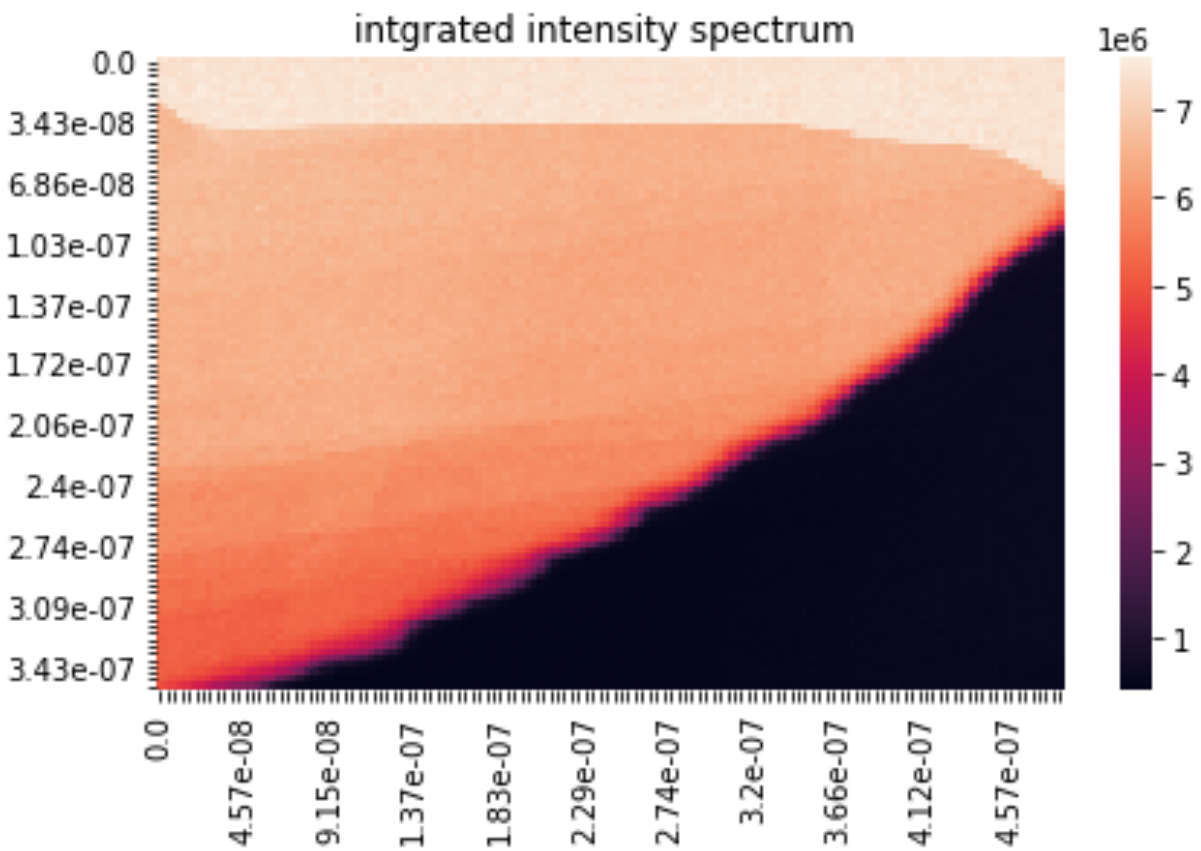
```
>>> im.crossings_im()
```

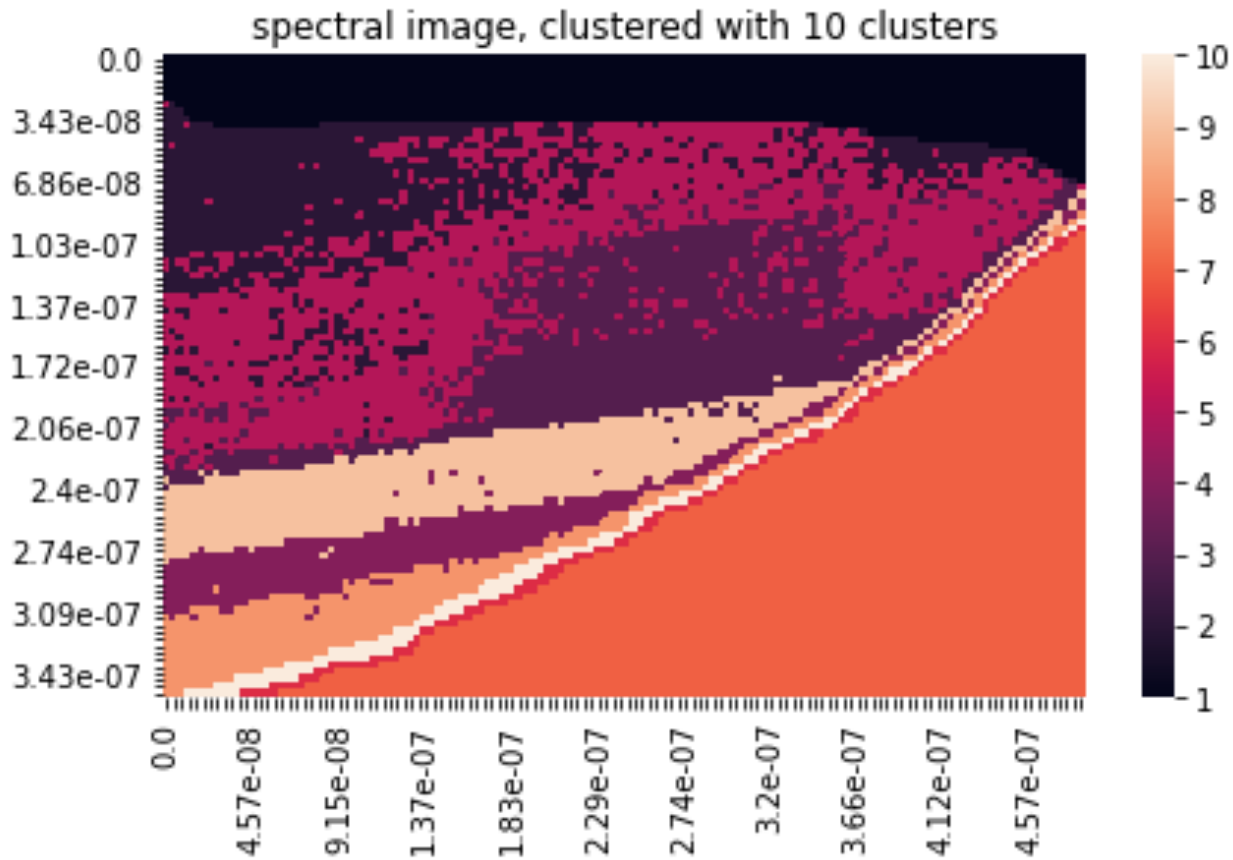
Start towards integrating ZLP NN-training

- K-means clustering, by default based on the integrated intensity, possible to add different differentiators

```
>>> im.cluster(n_clusters = k)
```

Different cluster outcomes





Newly clustered image