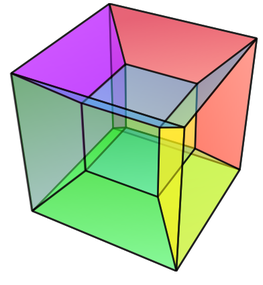
EDS-TEM quantification of core shell nanoparticles

- Using machine learning methods, the composition of embedded nanostructures can be accurately measured
- Demonstrated by D. Roussow et al., Nano Letters, 2015
    - See the full article for details
- Using the same data, this notebook reproduces the main results of this article



NIST
MATERIAL MEASUREMENT LABORATORY

Credits

- This notebook was originally written by Pierre Burdet in 2015, with subsequent edits by:
    - Duncan Johnstone — 2016
    - Francisco de la Peña — 2016
    - Pierre Burdet — 2016
    - Andy Herzing and Josh Taillon — 2018

- Requires HyperSpy v1.3+

1. Specimen & Data

The sample and the data used in this tutorial are described in D. Roussow, et al., Nano Letters, In Press (2015) (see the full article).

FePt@Fe_3O_4 core-shell nanoparticles are investigated with an EDS/TEM experiment (FEI Osiris TEM, 4 EDS detectors). The composition of the core can be measured with ICA (see figure 1c). To prove the accuracy of the results, measurements on bare FePt bimetallic nanoparticles from a synthesis prior to the shell addition step are used.

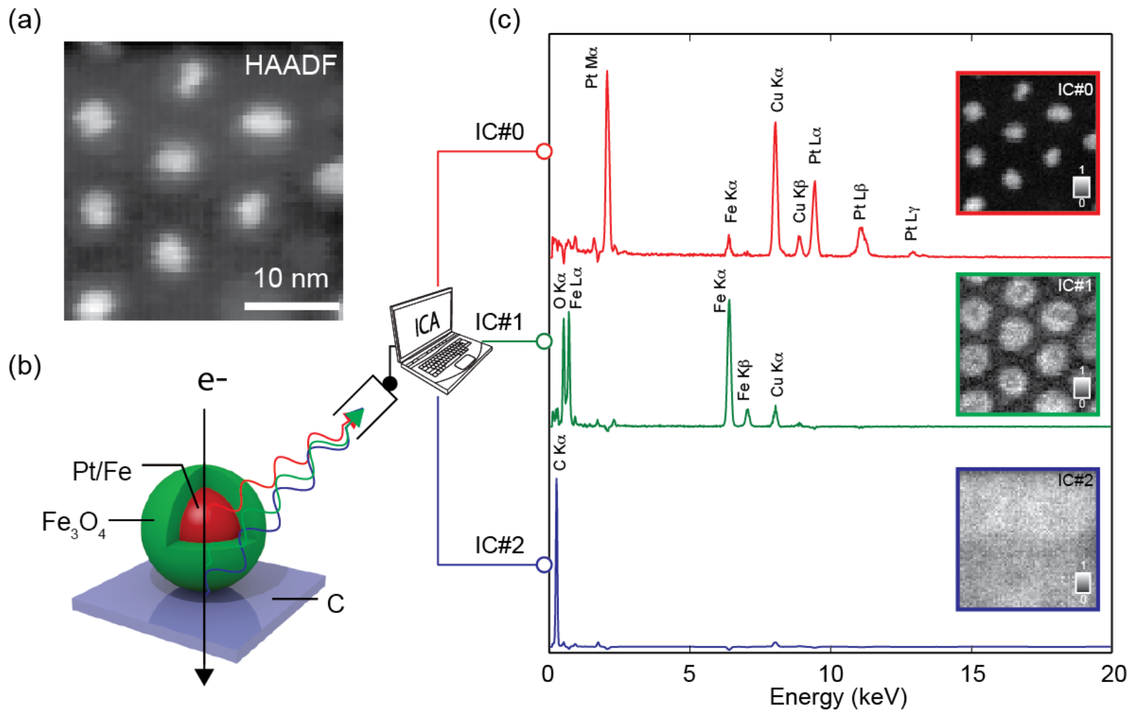


Figure 1: (a) A spectrum image obtained from a cluster of core-shell nanoparticles. (b) The nanoparticles are comprised of a bi-metallic Pt/Fe core surrounded by an iron oxide shell on a carbon support. (c) ICA decomposes the mixed EDX signals into components representing the core (IC#0), shell (IC#1) and support (IC#2).

2. Loading the data

Import HyperSpy, numpy and matplotlib libraries

In [2]:
```
%matplotlib nbagg
import hyperspy.api as hs
import numpy as np
```

Load the spectrum images of the bare seeds and the core shell nanoparticles.

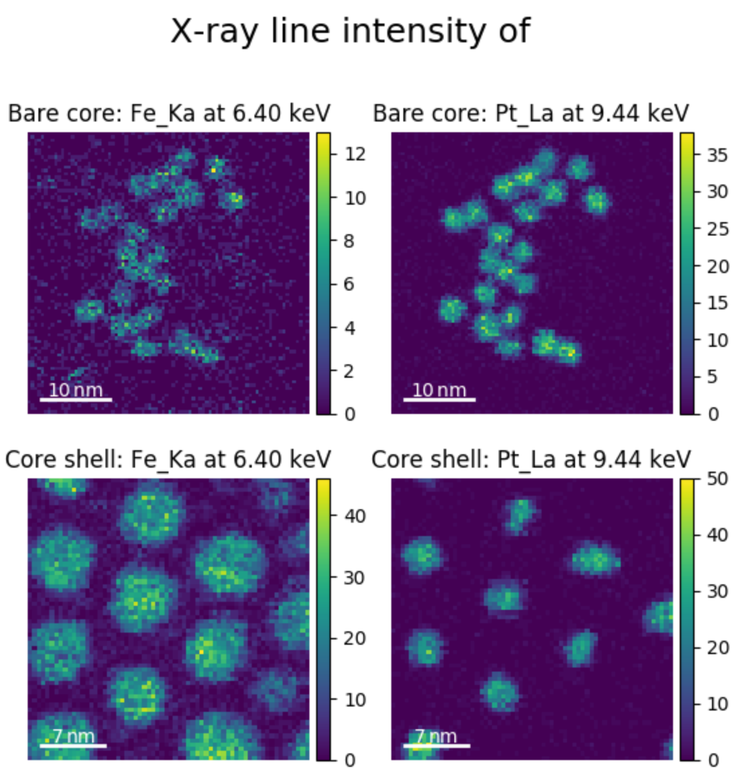
In [3]:
```
c = hs.load('examples/bare_core.hdf5')
cs = hs.load('examples/core_shell.hdf5')
```

In [4]:
```
# Examine the metadata of the cores:
c.metadata
```

Out[4]:
```
├── Acquisition_instrument
│   └── TEM
│       ├── Detector
│       │   └── EDS
│       │       ├── azimuth_angle = 0.0
│       │       ├── elevation_angle = 35.0
│       │       └── energy_resolution_MnKa = 130.0
│       ├── Stage
│       │   └── tilt_alpha = 0.0
│       ├── acquisition_mode = STEM
│       ├── beam_energy = 200.0
│       └── microscope = Microscope TecnaiOsiris 200 kV D658 AnalyticalTwin
├── General
│   ├── date = 14.10.2014
│   └── title = Bare core
├── Sample
│   ├── elements = array(['Fe', 'Pt'],
      dtype='<U2')
│   └── xray_lines = array(['Fe_Ka', 'Pt_La'],
      dtype='<U5')
└── Signal
    ├── binned = True
    ├── signal_origin =
    └── signal_type = EDS_TEM
```

Plot the intensity of Fe Kα and Pt Lα lines:

In [5]:
```
axes = hs.plot.plot_images(hs.transpose(*(c.get_lines_intensity() + cs.get_lines_intensity())),
                           scalebar='all', axes_decor='off', per_row=2, cmap='viridis')
```

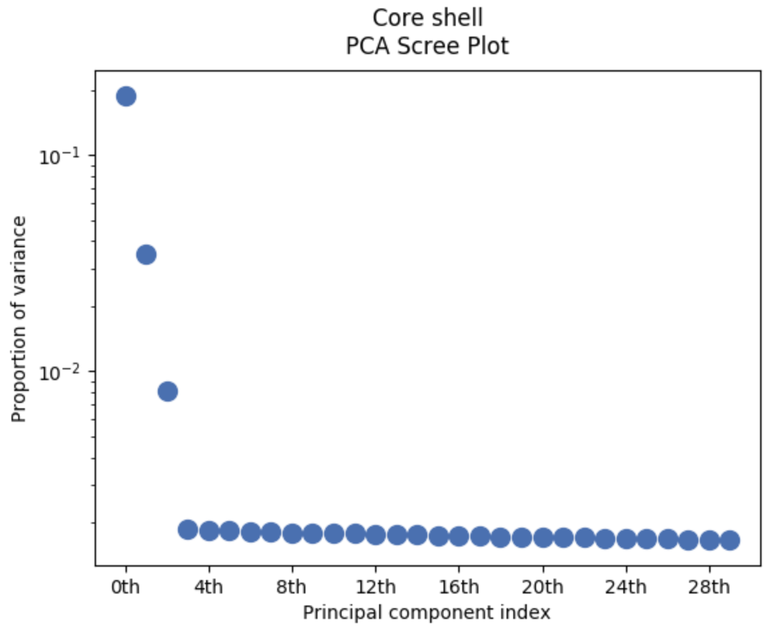


3. Blind source separation of core/shell nanoparticles

Apply blind source separation (ICA) to obtain a factor (spectrum) corresponding to the core.

In [6]:
```
# Have to change datatype to float for decomposition:
cs.change_dtype('float')
cs.decomposition()
```

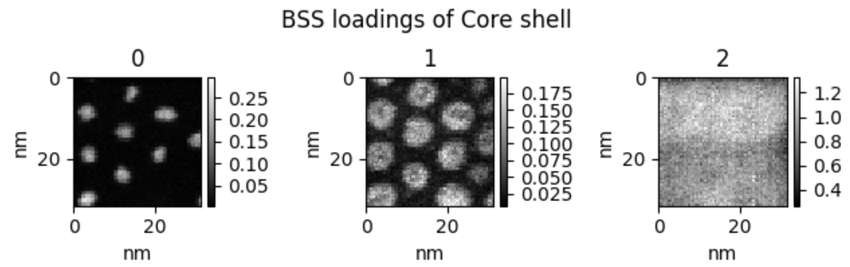
In [7]:
```
ax = cs.plot_explained_variance_ratio()
```



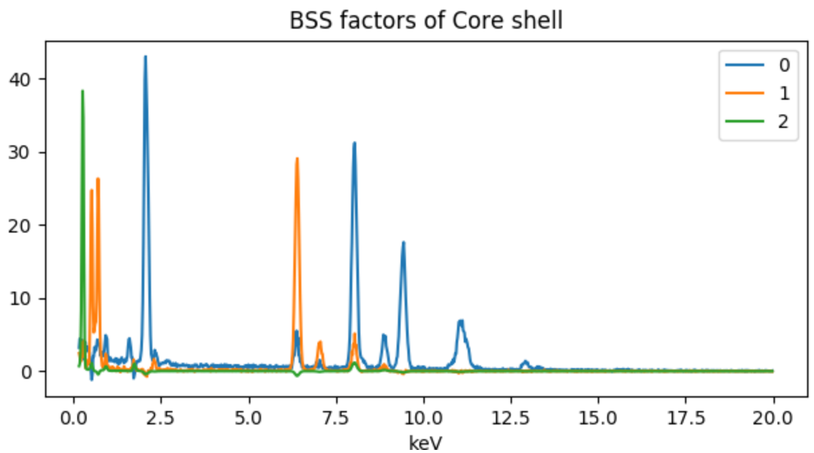
ICA on the three first components.

In [8]:
```
cs.blind_source_separation(3)
```

In [9]:
```
axes = cs.plot_bss_loadings()
```



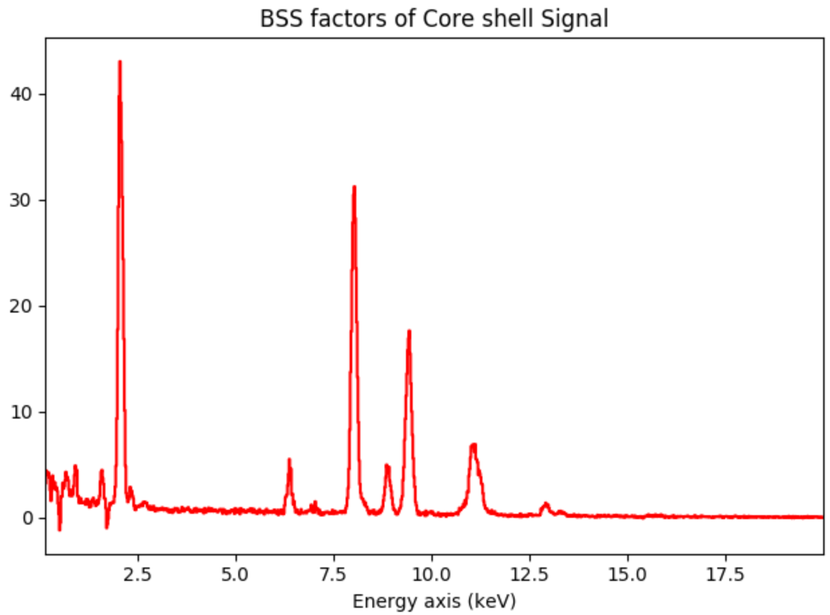
In [10]:
```
axes = cs.plot_bss_factors()
```



The first component corresponds to the core.

In [11]:
```
s_bss = cs.get_bss_factors().inav[0]
```

In [12]:
```
s_bss.plot()
```



4. Representative spectrum from bare cores

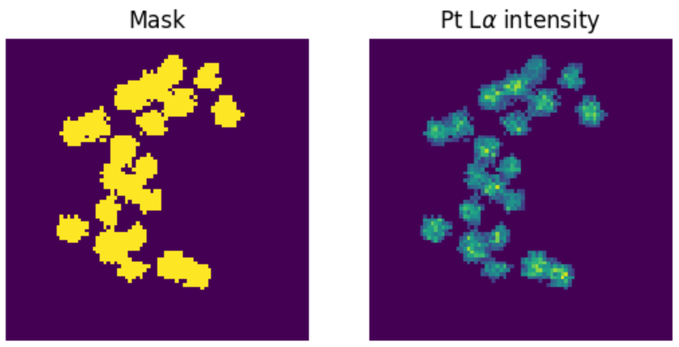
We meed to obtain a representative spectrum of the bare nanoparticles so we can compare to the BSS component

We can mask the low intensity of the Pt Lα signal:

In [13]:
```
pt_la = c.get_lines_intensity(['Pt_La'])[0]
mask = pt_la > 6
```

Visualizing the mask:

In [14]:
```
axes = hs.plot.plot_images(hs.transpose(*(mask, pt_la * mask)), axes_decor='off', colorbar=None,
                           label=['Mask', 'Pt L${\\alpha}$ intensity'], cmap='viridis')
```



Applying the mask:

- `mask` is a `Signal` containing boolean values, but it is 2D, not 3D:

In [15]:
```
print(mask.data.shape)
mask.data
```

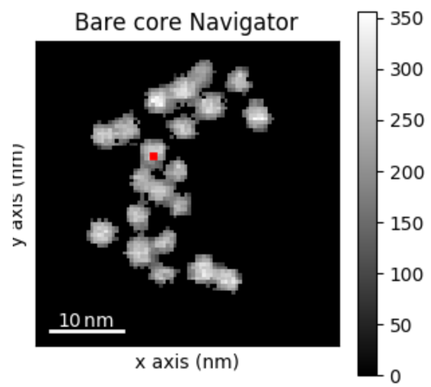
```
(84, 84)
```

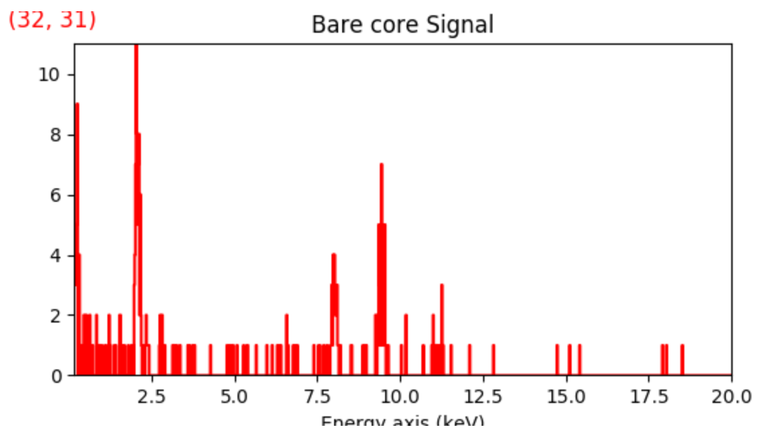
Out[15]:
```
array([[False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False]], dtype=bool)
```

- To apply the mask, we can just multiply the signals together thanks to `numpy`'s array broadcasting:

In [16]:
```
c_masked = c * mask
```

In [17]:
```
c_masked.plot()
```

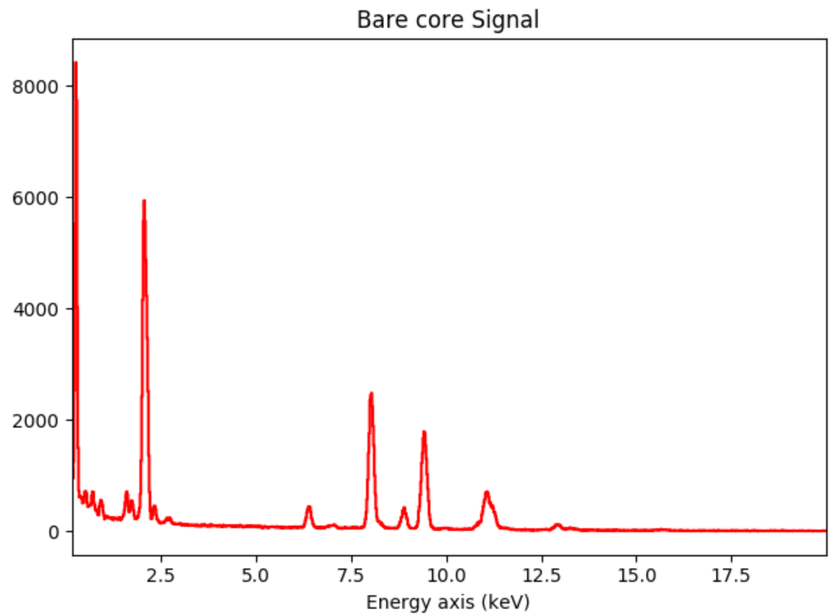




The sum over the masked particles is used as a bare core spectrum:

In [18]:
```
s_bare = c_masked.sum()
```

In [19]:
```
s_bare.plot()
```

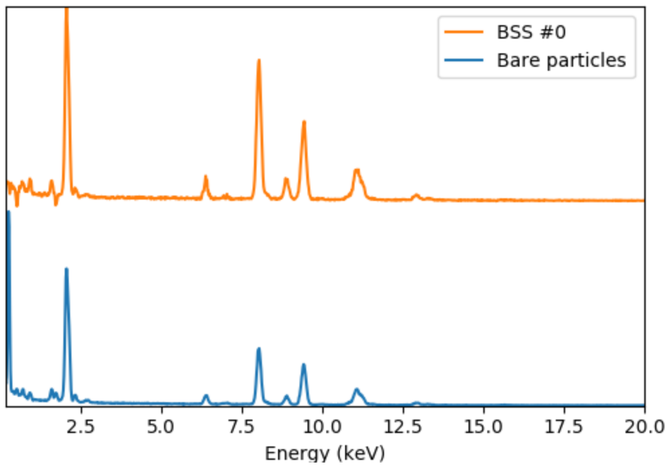


5. Comparison and quantification

We stack together the spectrum of bare particles and the first ICA component:

In [20]:
```
s_bare.change_dtype('float')
s = hs.stack([s_bare, s_bss], new_axis_name='Bare or BSS')
s.metadata.General.title = 'Bare or BSS'
```

In [21]:
```
# Normalize the data for plotting for easier comparison:
axes = hs.plot.plot_spectra([sig/sig.data.max() for sig in s], style='cascade',
                            legend=['Bare particles', 'BSS #0'])
```



Comparison method 1 — net intensity calculation

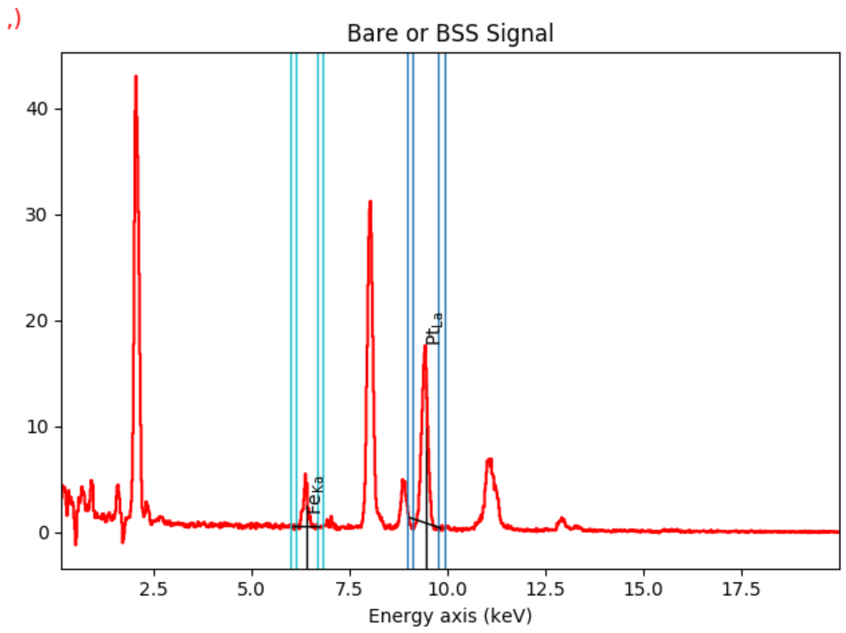
X-ray intensities measurement with background subtraction

In [22]:
```
w = s.estimate_background_windows()
```

In [23]:
```
s.plot(background_windows=w, navigator='slider')
```

Bare or BSS index 1 value 1

Continous update



Refinement of the windows position.

In [24]:
```
w
```

Out[24]:
```
array([[ 5.99958948,  6.13435965,  6.67344035,  6.80821052],
       [ 8.96061636,  9.1211109 ,  9.7630891 ,  9.92358364]])
```

In [25]:
```
w[1, 0] = 8.44
w[1, 1] = 8.65
w
```

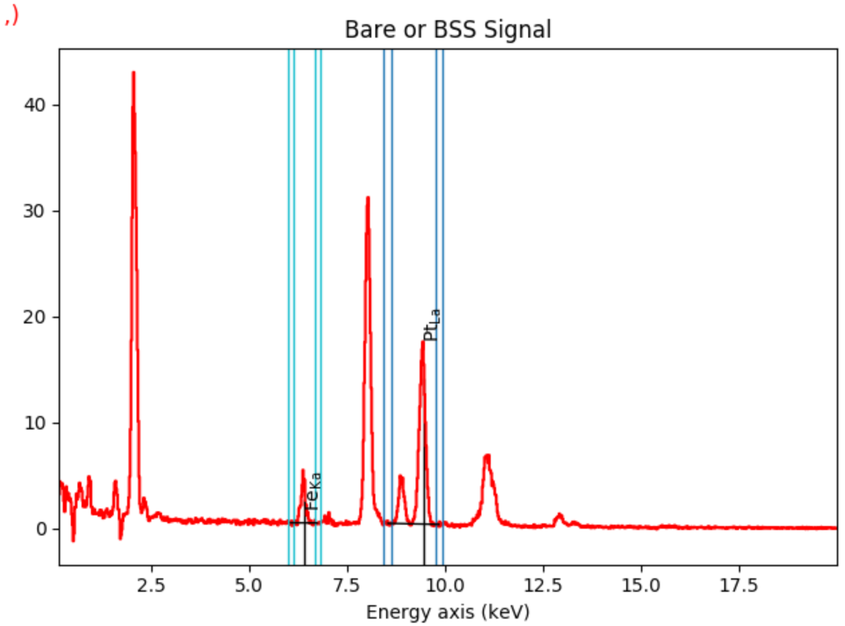
Out[25]:
```
array([[ 5.99958948,  6.13435965,  6.67344035,  6.80821052],
       [ 8.44      ,  8.65      ,  9.7630891 ,  9.92358364]])
```

In [26]:
```
s.plot(background_windows=w, navigator='slider')
```

Bare or BSS index 1 value 1

Continous update

,)



Get the net intensity under the peaks (using our new background windows):

In [27]:
```
sI = s.get_lines_intensity(background_windows=w)
sI
```

Out[27]:
```
[<BaseSignal, title: X-ray line intensity of Bare or BSS: Fe_Ka at 6.40 keV, dimensions: (2|)>,
 <BaseSignal, title: X-ray line intensity of Bare or BSS: Pt_La at 9.44 keV, dimensions: (2|)>]
```

Comparing the ratio of Fe intensity to Pt:

In [28]:
```
print('Bare core Fe_Ka/Pt_La: \t{:.2f}'.format(sI[0].data[0] / sI[1].data[0]))
print('BSS Fe_Ka/Pt_La: \t{:.2f}'.format(sI[0].data[1] / sI[1].data[1]))
```

```
Bare core Fe_Ka/Pt_La:  0.18
BSS Fe_Ka/Pt_La:        0.19
```

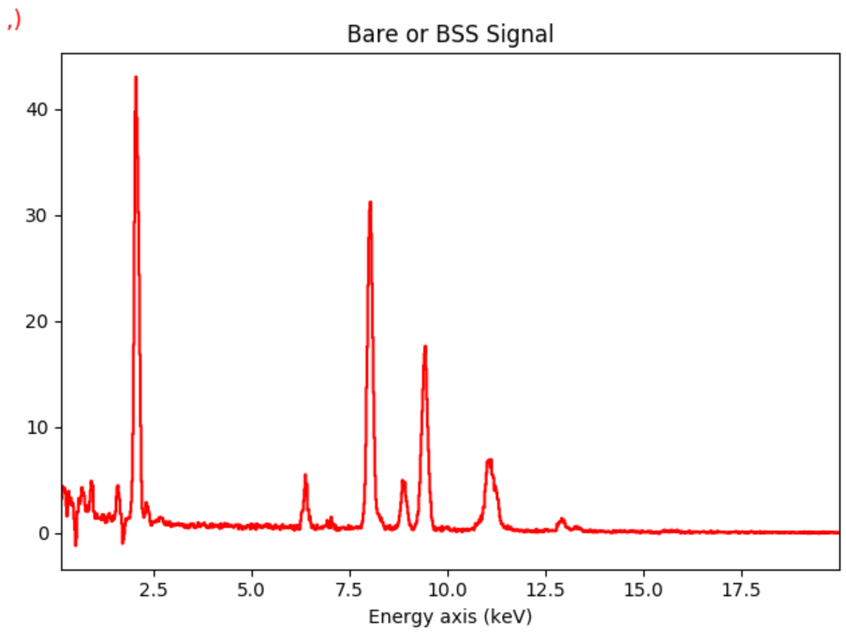
Comparison method 2 — model fitting

Measure X-ray intensity by fitting a Gaussian model

In [29]:
```
s.plot(navigator='slider')
```

Bare or BSS index 1 value 1

Continous update



In [30]:
```
# Create a model based off a cropped area of signal:
m = s.isig[5.:15.].create_model()
```

Add background copper and cobalt elements:

In [31]:
```
m.add_family_lines(['Cu_Ka', 'Co_Ka'])
```

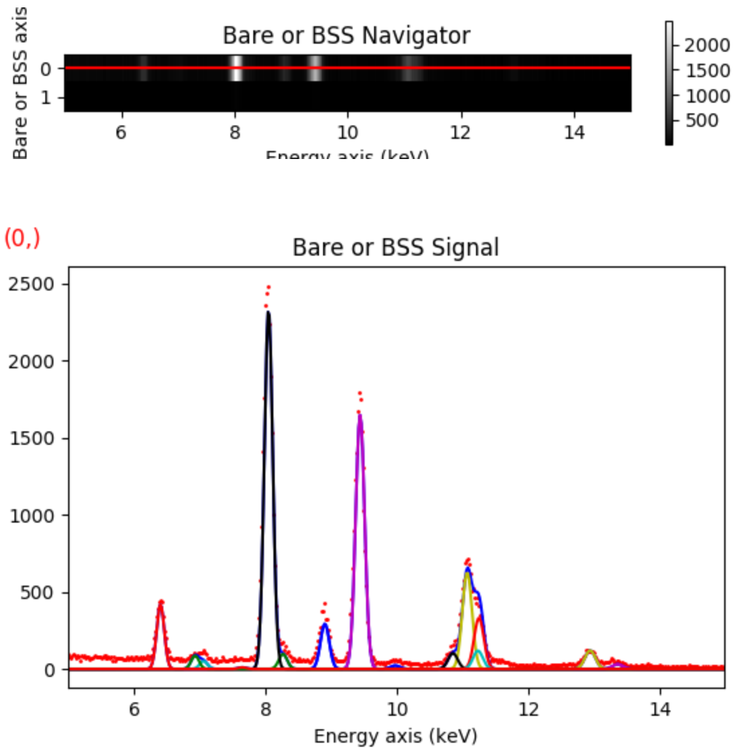
Contents of the model:

In [32]:
```
m.components
```

Out[32]:

#	Attribute Name	Component Name	Component Type
0	background_order_6	background_order_6	Polynomial
1	Fe Ka	Fe Ka	Gaussian
2	Fe_Kb	Fe_Kb	Gaussian
3	Pt_La	Pt_La	Gaussian
4	Pt_Lb1	Pt_Lb1	Gaussian
5	Pt_Lb4	Pt_Lb4	Gaussian
6	Pt_Ln	Pt_Ln	Gaussian
7	Pt_Ll	Pt_Ll	Gaussian
8	Pt_Lb2	Pt_Lb2	Gaussian
9	Pt_Lb3	Pt_Lb3	Gaussian
10	Pt_Lg3	Pt_Lg3	Gaussian
11	Pt_Lg1	Pt_Lg1	Gaussian
12	Cu_Ka	Cu_Ka	Gaussian
13	Cu_Kb	Cu_Kb	Gaussian
14	Co_Ka	Co_Ka	Gaussian
15	Co_Kb	Co_Kb	Gaussian

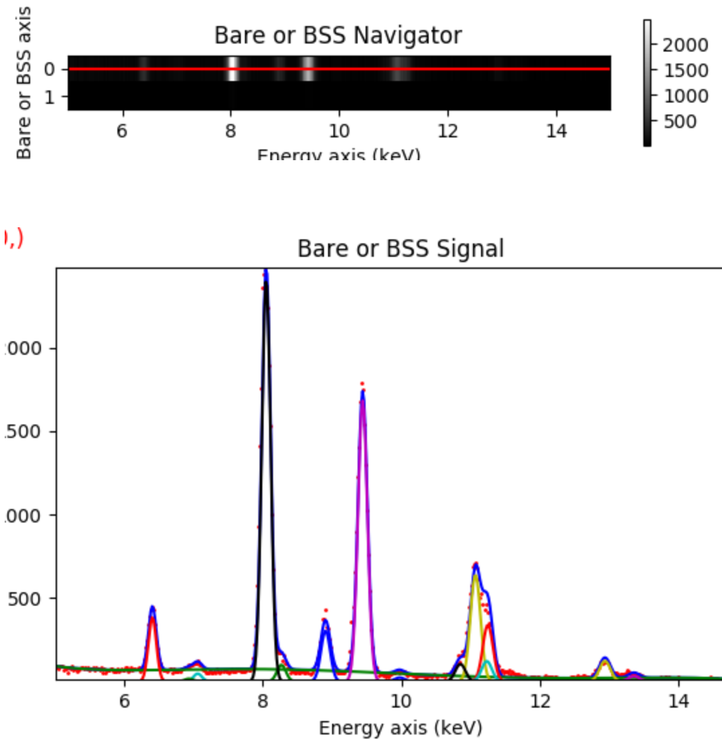
In [33]:
```
m.plot(plot_components=True)
```



Fitting the model at all locations of the signal is a simple one line command:

In [34]:
```
m.multifit(show_progressbar=False)
```

In [35]:
```
m.plot(plot_components=True)
```

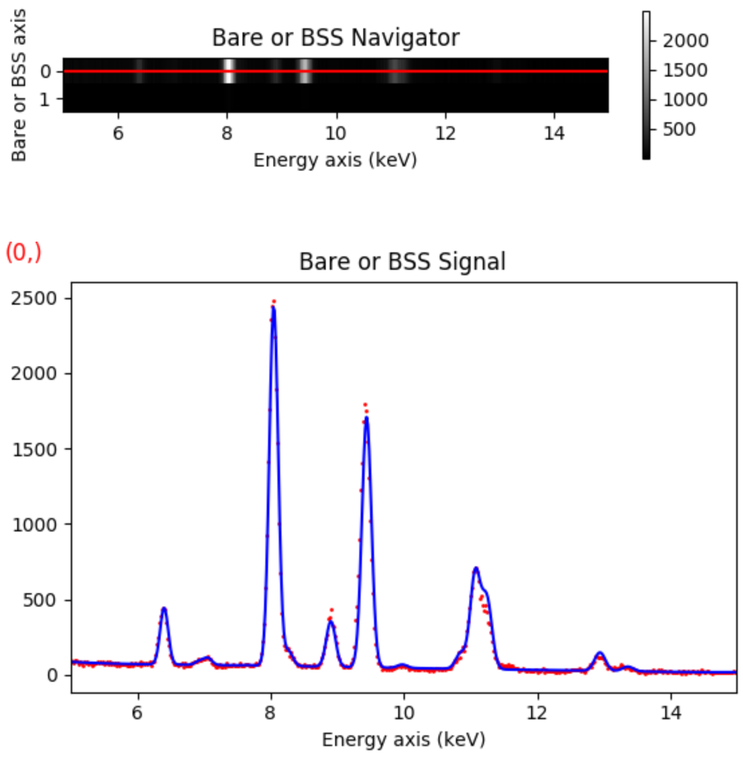


The background is fitted separately:

In [36]:
```
m.fit_background()
```

In [37]:
```
m.calibrate_energy_axis()
```

In [38]:
```
m.plot()
```



Finally, we probe line intensity from the fitted model:

In [39]:
```
sI = m.get_lines_intensity()[:2]
sI
```

Out[39]:
```
[<BaseSignal, title: Intensity of Fe_Ka at 6.40 keV from Bare or BSS, dimensions: (2|)>,
 <BaseSignal, title: Intensity of Pt_La at 9.44 keV from Bare or BSS, dimensions: (2|)>]
```

Set up the kfactors for Fe Kα and Pt Lα.

In [40]:
```
#From Bruker software (Esprit)
kfactors = [1.450226, 5.075602]
```

Quantify with Cliff Lorimer.

In [41]:
```
composition = s.quantification(method="CL", intensities=sI, factors=kfactors)
```

In [42]:
```
print('              |-----------------------------|')
print('              |     Atomic compositions     |')
print('              |-----------------------------|')

print(' \t      |  Bare core  |   BSS Signal  |')
print('|------------|-------------|---------------|')
print('| Fe (at. %) |    {:.2f}    |     {:.2f}     |'.format(composition[0].data[0], composition[0].data[1]))
print('| Pt (at. %) |    {:.2f}    |     {:.2f}     |'.format(composition[1].data[0], composition[1].data[1]))
print('|------------|-------------|---------------|')
```

```
              |-----------------------------|
              |     Atomic compositions     |
              |-----------------------------|
              |  Bare core  |   BSS Signal  |
|-------------|-------------|---------------|
| Fe (at. %) |    16.15    |     16.03     |
| Pt (at. %) |    83.85    |     83.97     |
|------------|-------------|---------------|
```

6. Going further

Further image processing with [scikit-image] and [scipy]. Apply a watershed transformation to isolate the nanoparticles.

- Transform the mask into a distance map.
- Find local maxima.
- Apply the watershed to the distance map using the local maximum as seed (markers).

Adapted from this scikit-image [example].

Import needed utilities from `scipy` and `skimage`:

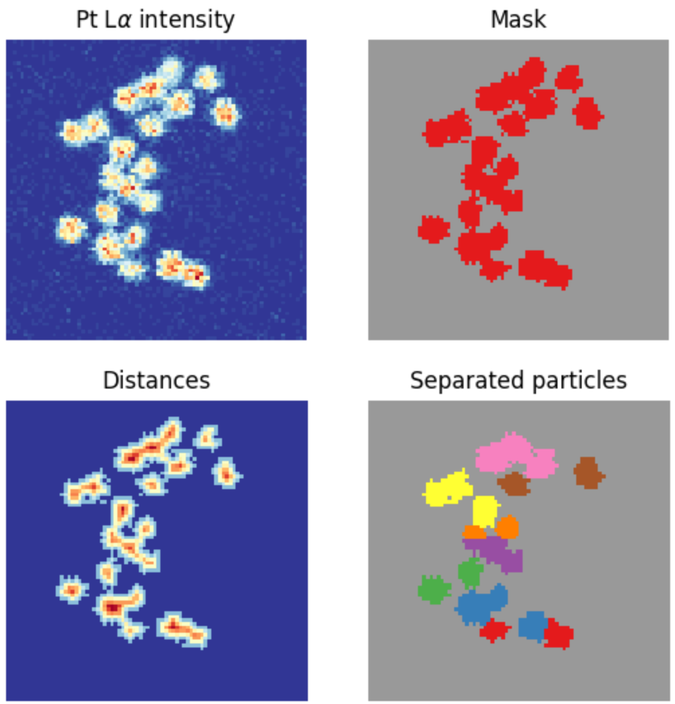
In [43]:
```
from scipy.ndimage import distance_transform_edt, label
from skimage.morphology import watershed
from skimage.feature import peak_local_max
```

Perform watershed segmentation:

In [44]:
```
distance = distance_transform_edt(mask.data)
local_maxi = peak_local_max(distance, indices=False,
                            min_distance=2, labels=mask.data)
labels = watershed(-distance, markers=label(local_maxi)[0],
                   mask=mask.data)
```

Plot the results:

In [45]:
```
axes = hs.plot.plot_images(
    [pt_la.T, mask.T, hs.signals.Signal2D(distance), hs.signals.Signal2D(labels)],
    axes_decor='off', per_row=2, colorbar=None, cmap=['RdYlBu_r', 'Set1_r'],
    label=['Pt L${\\alpha}$ intensity', 'Mask',
           'Distances', 'Separated particles'])
```



Questions?

Next demo: `tomotools`