Chapter IV

Introduction to hyperspectral image library in python (hsi)

Content

5. Introduction to python hyperspectral library

Hyperspectral imaging presents important current applications in medicine, agriculture, pharmaceutical, space, food and many upcoming applications. The analysis of hyperspectral images

requires advanced software. The upcoming developments related to fast hyperspectral imaging, automation and deep learning applications demand innovative software developments for analyzing hyperspectral data. Along this chapter a new software is development based on Python 3. The library is defined as hyperspectral imaging (*hsi*).

hsi is a python package for high level hyperspectral data processing scripting. The type of datasets can be any optical spectroscopic data with intensity and wavenumber axes. The upcoming content shows how to handle the data from an imaging spectroscopic point of view rather pure programming and developing approach. A high-level programming holds a short scripting structure for fast and efficient hyperspectral data analysis. *hsi* comes with 5 dataset-examples for understanding and practicing the tools implemented in *hsi*.

Table 5.1 datasets and examples

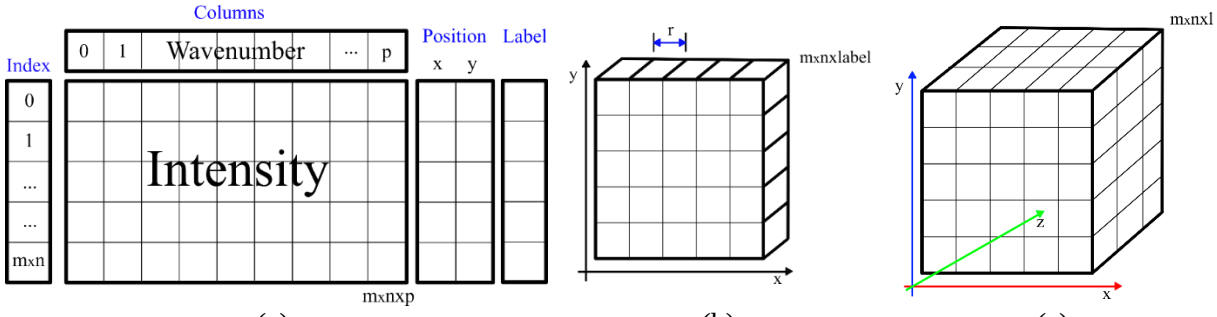
Plastics	Datasets of different plastics
Microplastics on tissue	Raman Map of tissue model and microplastics
Multi-layer of plastics	Z scanning of transparent plastic layers
Bladder	Cancer and noncancer tissue models
3D imaging	3D Raman Imaging of translucent tissue and plastics

The previous datasets will illustrate how to process raw datasets and obtain a processed result by several *hsi* tools. A complete information regarding *hsi* functions is shown in the appendix C.

5.1. Hyperspectral object structure (hsi)

Fig. 5.1 shows the hyperspectral data object that is stablished in the *hsi.py* package. The object holds an index for the spectra captured, columns hold wavenumber or *x* axis. The intensity table contains the data of the Raman intensity. Imaging datasets provide spatial information that are represented as *x* and *y* in position table. The *label* associates the meaning of the pixel of the image in one label name.

Hyperspectral imaging is represented by a 2D representation considering *x* and *y* coordinates and the information of label. Confocal Raman microscopy supplies *z* or depth information and thereby is possible to plot 3D images through voxels (also included in the library).



(a) (b) (c)

Figure 5.2. Hyperspectral object. (a) representation of hyperspectral object, (b) 2D plotting of hyperspectral data and (c) voxel illustration for 3D imaging.

5.2. Python, Anaconda and Spyder interface

The predetermined work interface is Spyder. For further information concerning how to install the required requirements, check the link: https://www.anaconda.com/. The required external libraries are: scikit-learn [1], standard libraries: matplotlib, NumPy and pandas, besides complex algorithms: vertex component analysis [2] and alternative iterative reweighted penalized least squares [3].

5.3. HSI objects, functions and assignments.

hsi is based on dataframe structure (Fig. 5.2) and contains properties (attributes) and methods for reading, manipulation, preprocessing, processing and visualization of hyperspectral data.

1. Download the *hsi* files that are in the link: https://github.com/darksiders123/hsi
2. Set the files in a working path.
3. Loading the hsi.py in the spyder console by selecting the path of the hsi files.
4. In spyder console read hsi library by: `from hsi import*`

5.3.1. Creating/Deleting an hsi object

Creating a hyperspectral instance/object. The hyper object can contain any kind of data, spc, csv, txt or custom as long as it keeps the standard dataframe shape (Fig. 5.2). The first methods are shown in the next lines.

Creating	`mapa = hyper_object('name')`
Deleting	`del mapa`

5.3.2. Reading/saving csv, spc, mat and any common file

Some standard data files are spc, csv and txt, the next lines show some reading methods.

Reading single spc	`mapa.read_single_spc(path)`
Reading several spc	`mapa.read_multi_spc(path)`
Reading a hologram map	`mapa.read_spc_holomap(path)`
Csv	`mapa.read_csv(path)`
Any file	`pandas.read_table(parameters) //(see example)`
Mat	`mat = read_mat(path)`
Show spectral data	`mapa.show(True), mapa.show(False)`
Save spectral data	`mapa.save_data(path, 'name')`

5.3.3. Basic Data Handling methods

This section provides methods for data manipulation such as getting intensity or position data. For example, the user can build the hyperobject using manually methods such as set_data() and set_position() and set_label(). The upcoming examples will explain how to use most of the following methods:

`data = mapa.get_data()`	`mapa.set_data(data)`
`position = mapa.get_position()`	`mapa.set_position(position)`
`wavenumber = mapa.get_wavenumber()`	`mapa.set_wavenumber(calibration_row)`
`number = mapa.get_number()`	`mapa.set_resolution(res)`
`label = mapa.get_label()`	`mapa.set_label(label)`
`name = mapa.get_name()`	`mapa.set_name(name)`
`number = mapa.get_number_spectra()`	`mapa.append(hyperobject)`
`spectrum = mapa.get_pixel(x, y)`	`mapa.reset_index()`
`Label = mapa.select_label(list)`	`mapa.concat([hyperobject1, ...])`
`Wavenumber = mapa.get_wavenumber()`	`mapa.set_wavenumber(wavenumber)`
`Copied = mapa.copy()`	`mapa.clean_data()`
	`mapa.remove_spectrum(index)`

	`mapa.rename_label(list_before, list_after)`
	`mapa.remove_label(list)`
	`mapa.add_peaks(0.5, 'red')`
	`mapa.remove_label([label])`
	`mapa.remove_spectrum([index])`

5.4. Pre-Processing

5.4.1. Selecting wavelength ranges

The selection of work range defines the spectral region of interest. Fig. 5.3 shows the full spectral range and examples of range selections.

Finger print region (Fig. 5.3(b))	`mapa.keep(lower = 400, upper = 1800)`
High wavenumber region (Fig. 5.3(c))	`mapa.keep(lower = 2800, upper = 3200)`
Without silent region (Fig. 5.3(d))	`mapa.keep(lower = 400, upper = 3200)` `mapa.remove(lower = 1800, upper = 2700)`

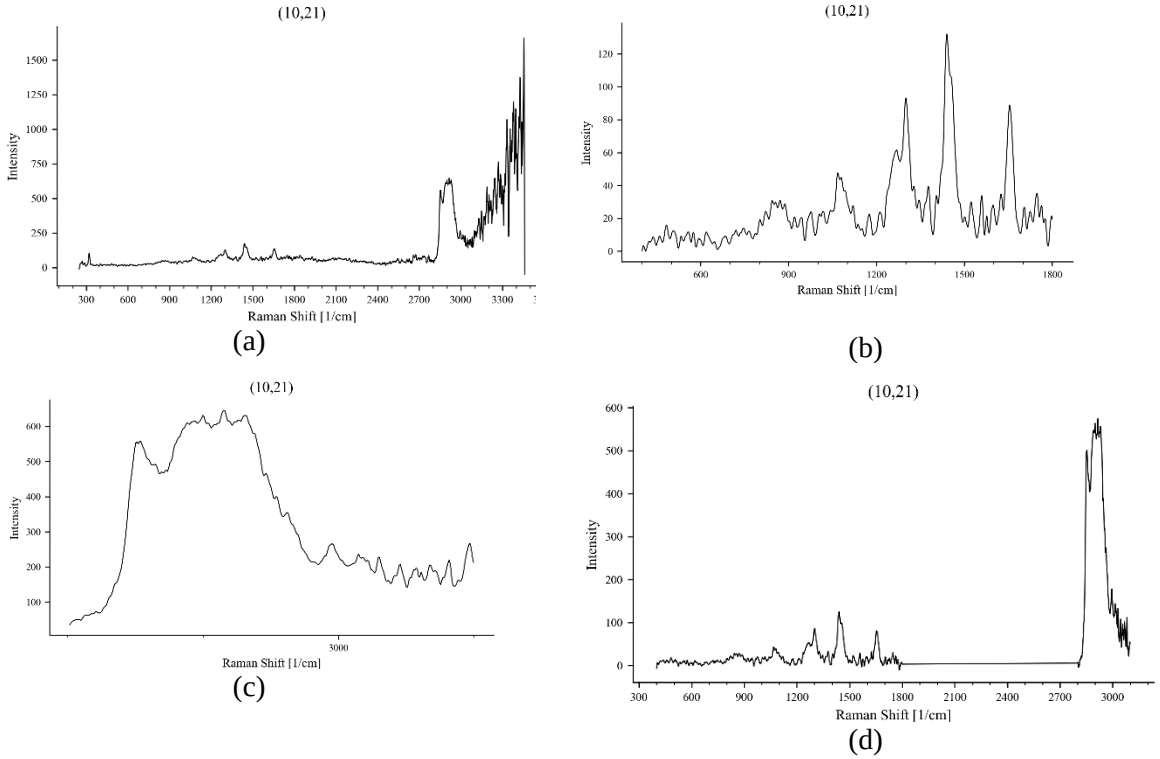


Figure 5.3. Selection of work range. (a) full range, (b) low wavenumber (finger print region), (c) high wavenumber and (d) selection of low and high wavenumber regions without the silent region.

5.4.3. Pre-processing

Methods for removing noise, outliers, baseline correction, normalization are presented in the next lines.

Removing Outliers

Intensity threshold	`mapa.threshold(lower = 50, upper = 500)`
Cosmic Ray removal tool	`mapa.spikes(threshold = 7, window = 3)`

Smoothing

Savitzky golay filter	`mapa.gol(window = 7, interpolation = 3, order = 0)`
Gaussian filter	`mapa.gaussian(variance = 2)`

Baseline correction

Rubber Band correction	`mapa.rubber()`
Snip baseline	`mapa.snip(iterations = 100)`
Alternative iterative reweighted partial least squares	`mapa.airpls(landa = 100)`

Normalization

Value / (Max - Min)	`mapa.norm()`
Considering a peak as reference	`mapa.norm_peak(peak = 1001)`
Vector normalization	`mapa.vector()`

5.5. Processing

Principal component analysis	`components, loadings = mapa.pca(num_components = 3 ,norm = 'False' )`
Partial least square and linear discriminant analsys	`components, loadings = mapa.pls_lda(num_components = 3 ,norm = 'False' )`
Vertex component analysis	`endmembers = mapa.vca(num_components = 3)`
Multi curve resolution	`components = mapa.mcr_als(spectra, constrain = 'NNLS' num_iterations = 100)`
Kmeans	`mapa.kmeans(3); mapa.kmeans('auto')`
Hierchical clustering	`mapa.hca('euclidean', 'ward', 2, 3)`
abundance	`Concentrations = mapa.abundance(endmembers, constrain = 'NNLS')`
dbscan	`Dbscan(min_samples, eps)`
hdbscan	`Hdbscan(min_samples, min_cluster)`

5.6. Visualization

The final product of processing is the correct visualization of the data by different methods such as plotting spectra, maps or scatter plots. The

Overlapped plot of spectra	`mapa.show_spectra(0,0,'auto')`
Stack of spectra	`mapa.show_stack(0,0,'auto')`
Profile	`mapa.profile('auto')`
Map	`mapa.show_map('auto', None, 1)`
Show scatter points	`mapa.show_scatter('auto', label, size)`
Show scatter points 3d	`mapa.show_scatter_3d('auto', label, size)`
Show 3d map	`mapa.scatter3D()`

5.7. Hyperspectral imaging examples

Set the work file path to where the specific file path is located.

5.7.1. Plastics

The following script shows how to use hsi library for reading some spectra data and apply baseline correction.

```
from hsi import*
### Defining paths ###
blue_path = 'examples/plastics/blue'
red_path = 'examples/plastics/red'
natural_path = 'examples/plastics/natural'
### Creating and reading the files ###
red = hyper_object('red') ## Declaring the first hyper object for red sample
red.read_csv(red_path) ## Reading the comma separated vector file
red.set_label('red') ## Setting red name to the whole data set spectra
red.keep(500, 1800) ## Keep finger region
meanred = red.mean() ## Compute mean of all red spectra

blue = hyper_object('blue') ## Same procedure for the blue sample
blue.read_csv(blue_path)
blue.set_label('blue')
blue.keep(500, 1800)
blue.rubber() ## Baseline correction rubber band
meanblue = blue.mean()

natural = hyper_object('natural') ## Same procedure for the natural sample
natural.read_csv(natural_path)
red.set_label('natural')
natural.keep(500, 1800)
meannatural = natural.mean()
### Concataneting the three hyperspectral objects
concat = hyper_object('concat') ## create a new empty object
concat.concat([meanred, meanblue, meannatural])
concat.show_stack(0,0,['red', 'blue', 'gray']) ## show the spectra (see Fig. 5.4)
```

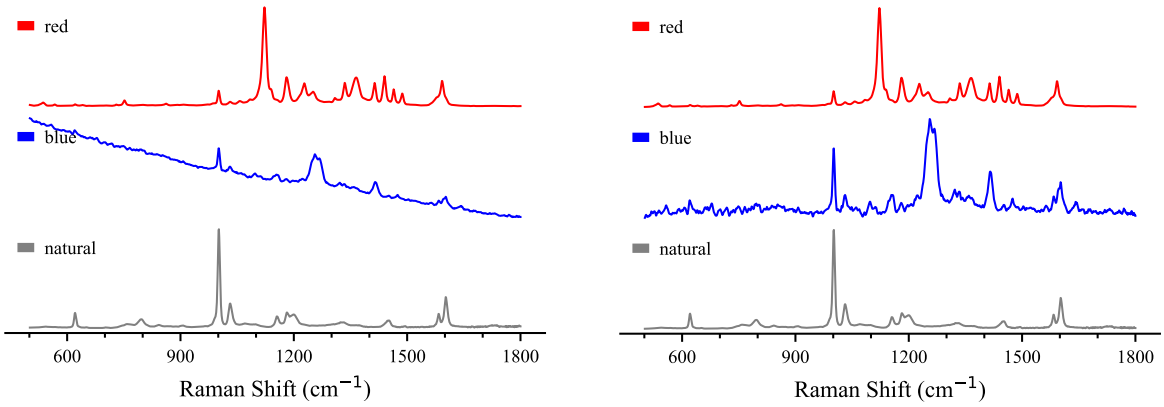
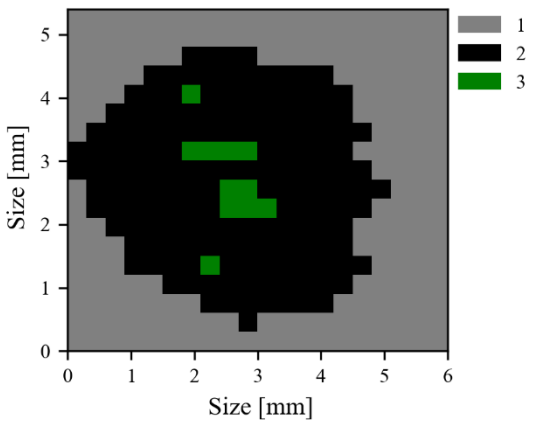


Figure 5.4. Modified plastics: (a) raw data and (b) final result

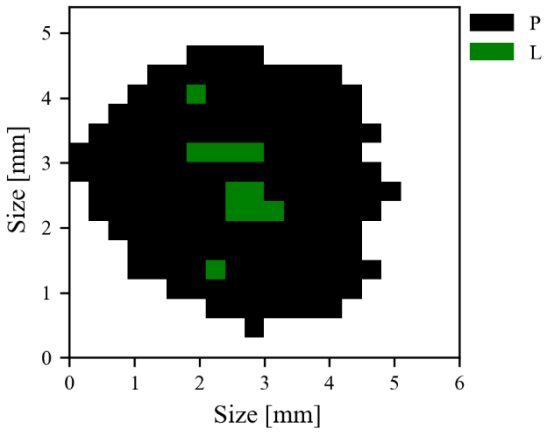
5.7.2. Bladder

The next example displays how to process a map image by clustering for obtaining the classification of different tissue

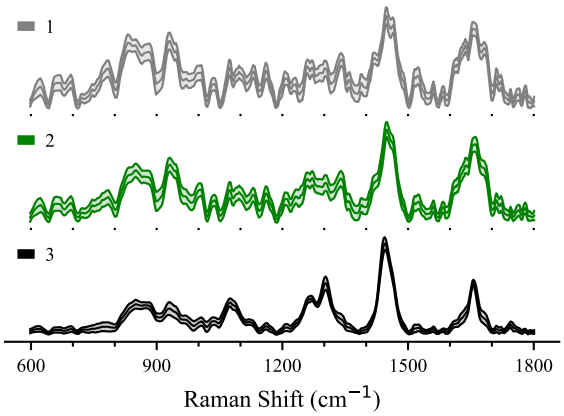
```
from hsi import*

### reading ###
path = 'H:/Thesis IPHT/Python/examples/bladder/data'
mapa = hyper_object('Patient')
mapa.read_csv(path)
mapa.set_resolution(0.3) ## 300 µm step size resolution
mapa.vector() # vector normalization
get_point = mapa.get_data() ## get data
### K-means clustering
mapa.kmeans(3) #K-means clustering: 3 components
colors = mapa.show_map(['gray', 'black', 'green'], None, 1) # show 2D map (Fig. 5.5(a))
mapa.show_stack(0, 0, colors) #show stack of the clusters (Fig. 5.5(c))
mapa.remove_label([1]) # remove the noisy cluster
mapa.rename_label([2, 3], ['P', 'L']) # rename the remaining clusters

colors = mapa.show_map(['black', 'green'], None, 1) # show 2D map (Fig. 5.5 (b))
mapa.show_stack(0, 0, colors) #show stack of the clusters

### Hierarchical clustering
mapa.set_data(get_point)
mapa.hca('euclidean', 'ward', 1.5, 3) # hca clustering with 1.5 distance  (Fig. 5.5(d))
colors = mapa.show_map(['gray', 'black', 'lightgray', 'green'], None, 1)
### Saving data
mapa.save_data(path, 'clustering') # saving data
```



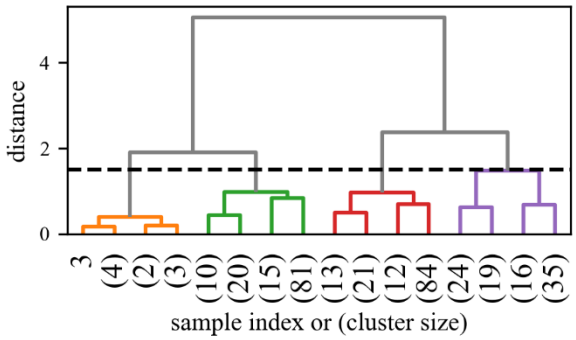
(a)



(b)



(c)



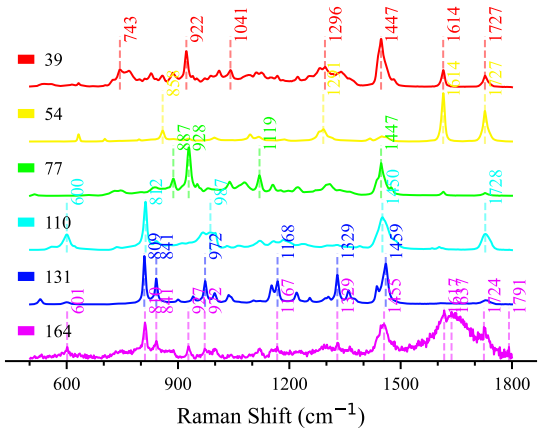
(d)

Figure 5.5. Clustering results: (a) K-means clustering map, (b) Clean clustering map (without background), (c) cluster means and (d) dendrogram of hierarchical clustering

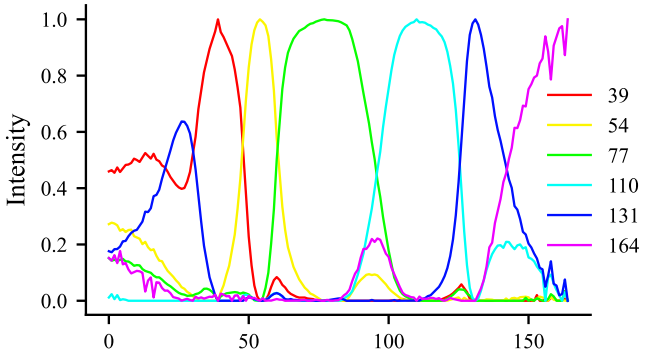
5.7.3. Multilayer

The following examples multi-layer of plastics illustrates how to analyze the data of a hyperspectral profile.

```
from hsi import*
### reading ###
path = 'examples/layers/data'
stack = hyper_object('layers') #creating the hyper object
stack.read_csv(path) #reading the csv file
### preprocessing ###
stack.keep(500, 1800) #finger print selection
stack.rubber() #baseline correction
stack.vector() #vector normalization
### processing ###
endmember = stack.vca(6) # vertex component analysis
endmember.show_stack(1, 0, 'auto') # visualization of spectra and strong
peaks(Fig.5.6(a))
abundance = stack.abundance(endmember, 'NNLS') # concentration estimation by NNLS
abundance.set_resolution(0.02) # set resolution of 20 µm for the profile
abundance.profile('auto') # plot profile (Fig.5.6(b))
```



(a)



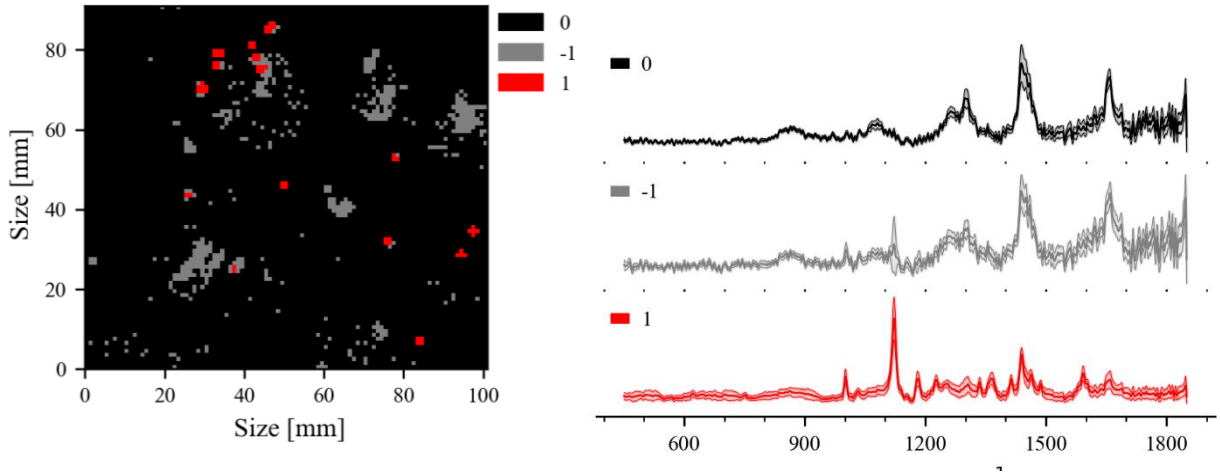
(b)

Figure 5.6. Hyperspectral profile analysis: (a) stack of plastics components found by VCA and (b) concentration profile.

5.7.4. Microplastic in tissue

The next examples illustrate the detection of pixel-sized microplastics on a tissue matrix using advanced clustering tools.

```
from hsi import*
### reading ###
path = 'examples/microplastic+tissue/data'
micro = hyper_object('MP') #creating the hyper object
micro.read_csv(path) #reading the csv file
### processing ###
micro.hdbscan(2, 15) # hierarchical density-based clustering
colors = micro.show_map(['gray', 'k', 'r'], None, 1) # 2D map of the clusters
(Fig.5.6(a))
micro.show_stack(0,0, colors) # stack of the spectral clusters (Fig. 5.6(b))
```



(a) (b)

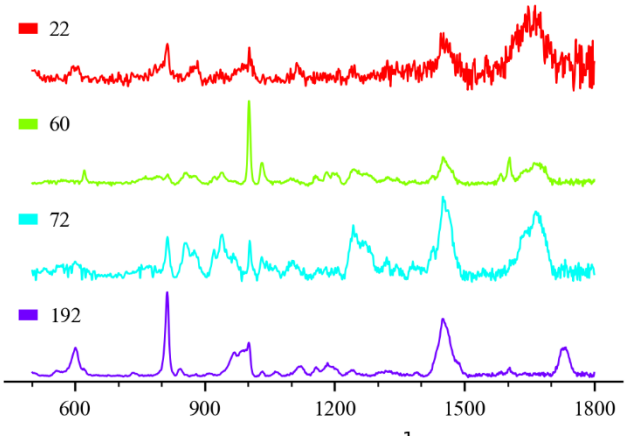
Figure 5.7. Density based clustering or detection of small pixels details: (a) hyperspectral image and (b) spectral cluster means.

3D hyperspectral imaging

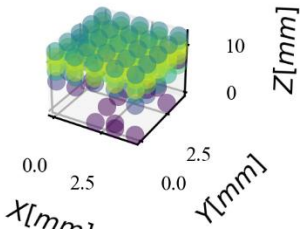
The final example shows how to handle basically 3D hyperspectral data and scatter 3D visualization as voxel.

```
from hsi import*
### reading ###
path = 'examples/3D/data'
cube = hyper_object('cube') #creating the hyper object
cube.read_csv_3D(path, 0.06, 0.07) #reading the 3d csv file and placing the
resolutions xy = 60 µm and z = 70 µm
### preprocessing ###
cube.keep(500, 1800) #finger print selection
cube.airpls(100) #advanced baseline correction
cube.vector() #vector normalization
### processing ###
```

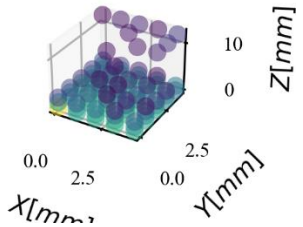
```
vca = cube.vca(4) # number of expected components
vca.show_stack(0, 0, 'auto')
abundance = cube.abundance(vca, 'NNLS') # concentration estimation by NNLS
aux = abundance.show_intensity_3d(0.3) # 3d plot of all clusters (Fig. 5.8(b-c))
```



(a)



(b)



(c)

Figure 5.8 3D imaging: (a) stack of the endmembers, (b) 3D concentration map of first layer 72 label and (c) 3D concentration map of 192.

References

[1] F. Pedregosa, G. Varoquaux, and A. Gramfort, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825-- 2830, 2011.

[2] J. M. P. Nascimento and J. M. B. Dias, “Vertex component analysis: A fast algorithm to unmix hyperspectral data,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, no. 4, pp. 898–910, 2005, doi: 10.1109/TGRS.2005.844293.

[3] Z. M. Zhang, S. Chen, and Y. Z. Liang, “Baseline correction using adaptive iteratively reweighted penalized least squares,” *Analyst*, vol. 135, no. 5, pp. 1138–1146, 2010, doi: 10.1039/b922045c.