SpMap: Hyperspectral package for spectroscopists in Python 3 Technical Document

Version 0.5

Juan David Muñoz-Bolaños

August 14, 2022

Contents

Juan David Muñoz-Bolaños[1,2,3*], Tanveer Ahmed Shaik[3], Ecehan Cevik[2,3], Shivani Sharma[2], Jürgen Popp[2,3] & Christoph Krafft[2]

[1] *Abbe School of Photonics, Albert-Einstein-Str. 6, 07745, Jena, Germany*
[2] *Leibniz Institute of Photonic Technology, Albert-Einstein-Str. 9, 07745, Jena, Germany*
[3] *Friedrich Schiller Jena University, Fürstengrabe 1, 07743, Jena, Germany*

**Corresponding author: jmunozbolanos@gmail.com*

1 Introduction

Hyperspectral imaging offers new applications in agriculture, pharmaceutical, space, food, and many upcoming applications such as medical diagnosis. The analysis of hyperspectral images requires advanced software for processing thousands of bands in the image. The forthcoming developments related to fast hyperspectral imaging, automation and deep learning applications demand innovative software developments. This manual presents a new package based on Python 3 for spectroscopists and enthusiasts for developing new applications in hyperspectral imaging. The package is defined as SpectraMap(SpMap).

SpMap is a python package for high level hyperspectral data processing and scripting. The type of datasets can be any optical spectroscopic data with intensity and wavenumber axes. For example, infrarred and Raman spectroscopies.

The upcoming content shows how to handle data from an imaging spectroscopic point of view rather pure programming and developing approach. A high-level programming software offers a short scripting structure for fast and efficient hyperspectral data analysis. SpMap comes with several and research dataset-examples for understanding and practicing the tools implemented in SpMap.

The Table 1 shows examples to explore and use the tools in SpMap. A complete information regarding SpMap methods are shown throughout this manual.For the data related to the examples visit examples.

1.1 Hyperspectral object structure

Fig. 1 shows the hyperspectral data object structure that is established in the SpMap package. The object holds an index and columns along x-axis or wavenumber. The rows contain the Raman intensity (Fig. 13a). Datasets

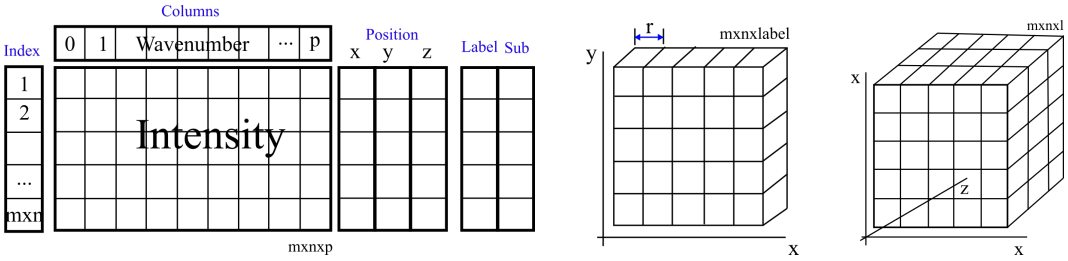
Table 1: Examples in SpMap

No.	Example	Description	Page
1	Raman spectra of plastics	Read CSV file, baseline correction and visualization of Raman data.	19
2	Depth profiling of multilayer of plastics	Read CSV, preprocessing and depth profile analysis of Raman data.	20
3	Microplastics in tissue	Detection of microplastics in tissue.	21
4	3D Raman hyperspectral image	Detection of microplastics in tissue by 3D analysis.	22
5	Tumor in bladder sections	Segmentation of biomolecues.	23
6	Infrarred hyperspectral image	Analysis of an infrared image.	23
7	Hyperspectral images of bladder	Pre and processing of bladder Raman images	24
8	Plastics: unmixing by PCA and PLS-LDA	Separation of plastics by principal components and linear-discriminant analysis	27

from 2D or 3D imaging provide spatial information represented as x, y and z in the position table of every pixel. The label is associated to the naming and meaning of every pixel. Sublabel offers an extra labeling useful for plotting and tracking changes. Confocal Raman microscopy supplies also z or depth information to plot 3D images Fig.1c .

1.2 Python and installation of SpMap

The predetermined work interface is Python 3. For further information concerning how to install the necessary requirements, visit the link SpMap PyPi. The required external libraries are: Scikit-Learn [1], standard libraries: matplotlib, NumPy and pandas, besides complex algorithms: vertex component analysis [3], alternative iterative reweighted penalized least squares [5] and hierarchical density based clustering with applications with noise HDBSCAN [2]. After the installation of SpMap, we can import SpMap in Python:



(a) (b) (c)

Figure 1: Hyperspectral Object, (a) visual representation, (b) 2D map, (c) 3D volume

```
# importing SpMap library in python
from spectramap import spmap as sp
```

1.3 SpMap hyperobject, methods and attributes

SpMap is based on dataframe structure (Fig. 13a) that contains attributes for containing the data of intensities, wavenumber, label and positions and methods for reading, manipulation, preprocessing, processing, and visualization of hyperspectral data. Visit the link: https://github.com/spectramap/spectramap for further information and to find examples.

1.4 Basic Data Handling methods

1.4.1 Creating/Deleting an SpMap object

The hyper object can contain any kind of data, spc, csv, txt or custom if it keeps the standard dataframe shape (Fig. 13a). The next command lines show how to create and delete the hyperobjects:

```
# creating an hyperobject called name in the variable mapa
mapa = sp.hyper_object('name')

# deleting mapa hyperobject
del mapa
```

1.4.2 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. The variable path may be the directory path and name for single readings and for multi readings path is the directory file of the datasets.

```
# Reading single spc file
mapa.read_single_spc(path)
# Reading several spc files
mapa.read_multi_spc(path)
# CSV (standard file in SpMap)
mapa.read_csv(path)
# compressed CSV (standard file in SpMap)
mapa.read_csv_xz(path)
# Any file (custom reading) (see coming example)
pandas.read_table(parameters)
# Mat (read a raw Mat) (require custom reading)
mat = read_mat(path)
# Show spectral data, average and individual
mapa.show(True), mapa.show(False)
# Save spectral data (CSV)
mapa.save_data(path, 'name')
# Save compressed spectral data (CSV_xz)
mapa.save_data_xz(path, 'name')
```

1.4.3 Other 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(data) and setposition(position) and setlabel(label). The upcoming examples will explain how to use most of the

Non returning methods

The next methods do not return any data.

```
# set a Pandas DataFrame as intensity (data)
mapa.set_data(data)
# set a Pandas DataFrame as position (xy)
mapa.set_position(position)
# set a Pandas Series as wavenumber
mapa.set_wavenumber(calibration)
# set the resolution (pixel size )in µm
mapa.set_resolution(res)
# set label name to the pixels
mapa.set_label(label)
# set sublabel name to the pixels
mapa.set_sublabel(sublabel)
# set name of the hyperobject
mapa.set_name(name)
```

```
# append another hyperobject
mapa.append(hyperobject)
# reset index enumeration
mapa.reset_index()
# concatenation of hyperobjects
mapa.concat([hyperobject1, ...])
# remove specific spectrum/pixel at index
mapa.remove_spectrum([index])
# clean data according defined label pixels
mapa.clean_data()
# clean position according defined data pixels
mapa.clean_position()
# clean label according defined data pixels
mapa.clean_label()
# rename labels by two lists (before and after)
mapa.rename_label([before], [after])
# remove specific label name by a list
mapa.remove_label([list])
```

Returning methods

The next methods return a type of data, hyperobject or DataFrames.

```
#Obtaining intensity
data = mapa.get_data()
#Obtaining positiion
position = mapa.get_position()
#wavenumber
wavenumber = mapa.get_wavenumber()
#copy of the original hyperobject
copied = mapa.copy()
# label
label = mapa.get_label()
# sublabel
label = mapa.get_sublabel()
# name
name = mapa.get_name()
# selection specific label
label = mapa.select_label(list)
# show and get wavenumber positions of the peaks
peaks = mapa.show2(True, prominence, color)
```

2 Raman Calibration

Reproducibility and replicativity are guaranteed by a appropriate calibration. A common method for wavenumber calibration is shown in this section. There are different materials for the calibration, such as Neon, Silica, Polystyrene or Paracetamol. In the next example will use Paracetamol and a laser 785 nm.

2.1 Wavenumber calibration

The basic idea is to measure a raw spectrum with the uncalibrated spectrometer and then with well-known spectrum compare the peak position between the pixels of the camera (uncalibrated spectrum) and the known spectrum (calibrated peaks along wavenumber). The next code shows how to calibrate a spectrometer. The function wavenumbercalibration requires special parameters carefully chosen. The error of calibration must be larger than 0.999. Fig. 16 shows the raw spectrum in pixels, the fitting and the calibrated spectrum.

```
from spectramap import spmap as sp # reading spectramap library
### Paracetaminol
mp = sp.hyper_object("para")
#read data with columns in pixels
mp.read_csv_xz("para")
copy = mp.copy() # copy data
# finding peaks of para (next plot)
peaks = copy.calibration_peaks(mp, 0.05)
# determining regression for the calibration
copy.wavenumber_calibration(peaks)
# set the new wavenumber to the original
mp.set_wavenumber(copy.get_wavenumber())
mp.keep(300, 1850)
mp.show2(True, 0.1, "r") # add peaks (not inline mode)
```

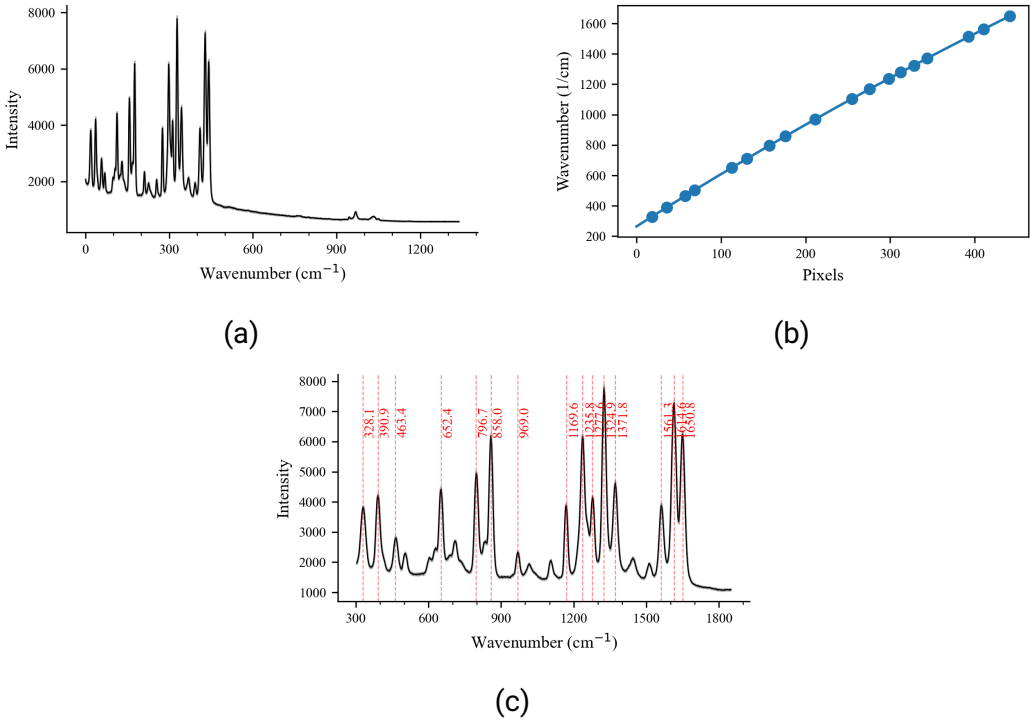


Figure 2: Wavenumber calibration, (a) paracetamithen spectrum in pixels, (b) calibration and (c) spectrum calibrated

2.2 Intensity calibration

After a proper wavenumber calibration is also required a intensity calibration. The calibration is done by a reference lamp, a measured intensity with the same lamp. The next code shows the calibration with an example of tissue spectrum calibration. Fig. 9 illustrates the calibration with idean and measured lamp spectra and sample spectrum.

```
# creating reference hyper object
reference_trial = sp.hyper_object("reference")
# reading the lamp referece data along wavelength
reference_trial.read_single_spc("reference")
# showing the spectrum in the next plot
reference_trial.show(True)
# creating hyper object
measured_trial = sp.hyper_object("measured")
# reading lamp experimental data
measured_trial.read_single_spc("lamp")
# keeping finger print region
```

```
measured_trial.keep(400, 1900)
# showing the plot as the next figures shows
measured_trial.show(True)

sample = sp.hyper_object("sample")
sample.read_single_spc("sample")
sample.keep(400, 1900) # finger print region
# intensity calibration function
sample.intensity_calibration(reference_trial, measured_trial)
sample.show(True) # showing the calibrated data in the next figure
```

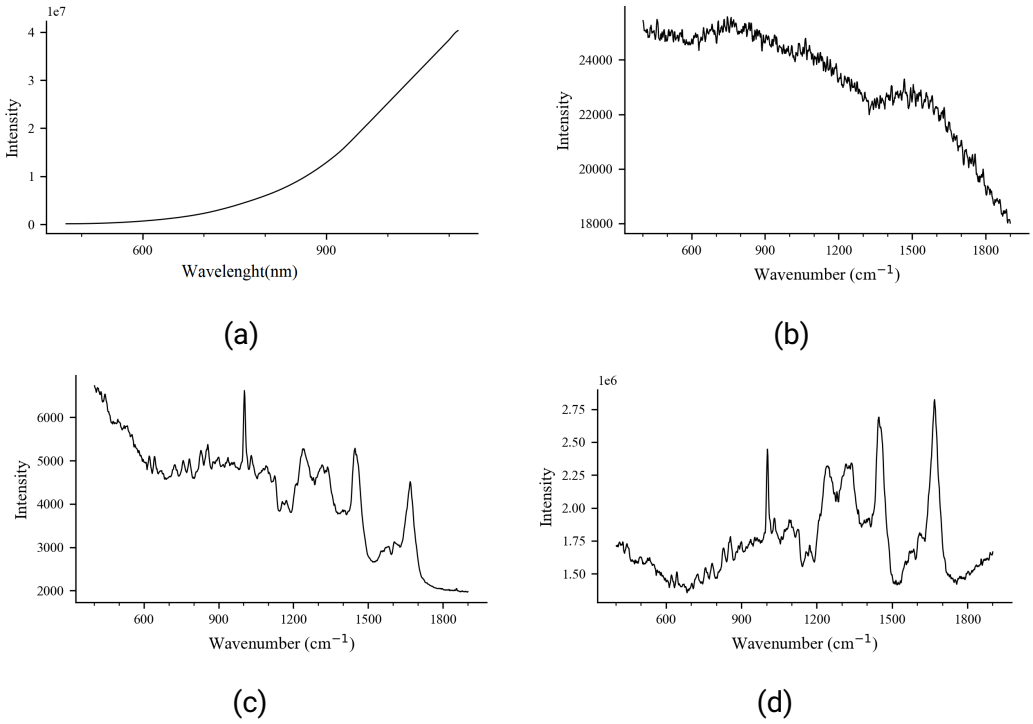


Figure 3: Intensity calibration, (a) reference intensity, (b) measured intensity, (c) uncalibrated sample and (d) calibrated sample.

3 Processing hyperspectral data

Hyper-spectroscopic measurements depend on hundreds of variables (wavenumbers) and observations (samples). Multivariate analysis figures out relationships between variables and observations. Several data mining approaches appear for this purpose such as principal component analysis (PCA), VCA (vertex component analysis), supervised cluster analysis (CA)

and unsupervised. The analysis procedure holds preprocessing the data to remove noise and improve the signal quality and then processing the data by supervised or unsupervised methods. Baseline correction, remotion of outliers and smoothing are common preprocessing steps.

3.1 Pre-Processing

After a proper collection of data or series of measurements. the next step is preprocess the data.

3.2 Substraction: background

First step in most of instruments is to remove the background noise. In Raman is collected a spectrum without any sample and with the laser on. The further measurements with samples must subtract the background spectrum.

3.3 Wavenumber intervals

The instrument configuration may offer different ranges of spectroscopic data. It is defined two main regions, low wavenumber (finger print region) and high wavenumber range. The selection of an appropriate range for processing is crucial since the samples behave different.

3.4 Filters

Hyperspectral filters may smooth noisy data and make a more homogeneous dataset.

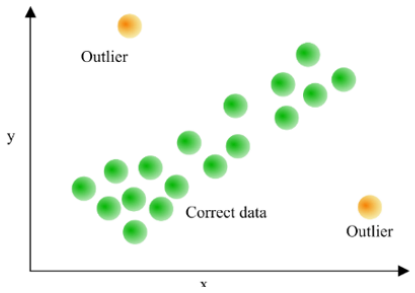
3.4.1 Savitzky-Golay Filter (gol)

Based on local least-squares polynomial approximation. It is a lowpass filter and reduce noise while it keeps the shape and height of the signal. Savgol preserves the higher peaks better than averaging methods. A signal with N measured points and a filter width (window), w, savgol calculates a polynomial fit of order k in each window as the filter moves across the signal [4].

3.4.2 Outlier removal (threshold)

Usually, some values in the dataset are far away from the expected position of values, these values are considered as outliers points. They are instrumental

artifacts and sample variations. One effective way to remove outliers is by noise-signal ratio thresholds.



(a)

3.5 Baseline corrections

SpMap offers three baseline corrections: rubber band, SNIP and adaptive iteratively reweighted penalized least square (airPLS).

3.5.1 airPLS

Adaptive iteratively reweighted penalized least squares (airPLS). The algorithm is an interactively reweighted procedure for finding gradually a complex baseline by minimizing sum squared derivatives (SSD) between a previous fitted baseline and original signal. The penalized least square algorithm is a smoothing algorithm. The user does not need to know prior information for the adjusting and minimization, this makes the algorithm a general baseline correction for hyperspectral datasets with wide diversity of spectra. For instance, polynomial corrections depend on user's experience and have poor development for low signal-to-noise. The airPLS solves iteratively a weighted penalized least squares problem as eq (1.9) presents [5].

$$Q^t = \sum_{i=1}^{m} w_i^t |x_i - z_i^t|^2 + \lambda \sum_{i=2}^{m} |z_j^t - z_{j-1}^t|^2 \tag{1}$$

Q measures the fidelity (eq. 1 first term) and smoothness (second term), w is the weight vector of fidelity, x is the unprocessed data and z is the baseline and λ is adjusted by the user, t is the number of iterations.

3.6 Normalizations

Different instruments, time, environment conditions and laser power yield intensity fluctuations. Normalization adjusts the intensity values to certain known range. The vector S (eq. 2 is a spectrum and SN (eq. 3) is a normalized spectrum. N is the number of spectral data points.

$$S = (s_1, s_2, \dots, s_N) \tag{2}$$

$$SN = (sn_1, sn_2, \dots, sn_N) \tag{3}$$

One common method of normalization in spectroscopy is vector normalization, which is defined by eq. 4 and 5. The norm is the square root of every square intensity value of spectrum and the norm is used for vector normalization.

$$norm = \sqrt{s_1^2 + s_2^2 + \cdots + s_N^2} \tag{4}$$

Each Raman intensity point is divided by the 'norm' to obtain the vector normalization result

$$sn_i = \frac{s_i}{norm}; i = 1, 2, \dots, N \tag{5}$$

There are several normalizations methods such maximum peak intensity, area or standard normal variate normalization (SNV).

4 Processing

Multivariable analasys is the core of hyperspectral processing. Three types of analysis exist: unsupervised, supervised and deep neuronal networks.

4.1 Unmixing

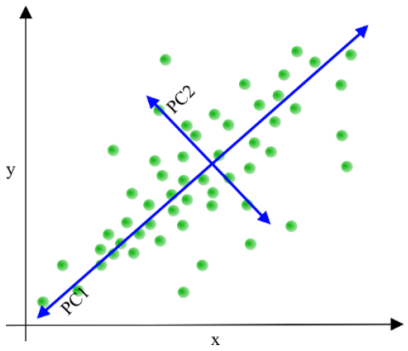
4.1.1 Principal components analysis (PCA)

PCA is an unsupervised method that transforms a large multidimensional data into lower number of dimensions. PCA reduces the data by finding orthogonal and independent components where the maximal variation stands out. Generally, the first components are most important and contain the main information while the last components hold noise significantly. Some methods for determining the suitable number of components are scree plots and percentage of variance. The transformation can be presented by eq. 6.

The loading matrix UK contains the k principal components and works as a transformation matrix between the original coordinate and new PC system.

$$X = YU_K^T + E$$
$$Y = XU_K \tag{6}$$

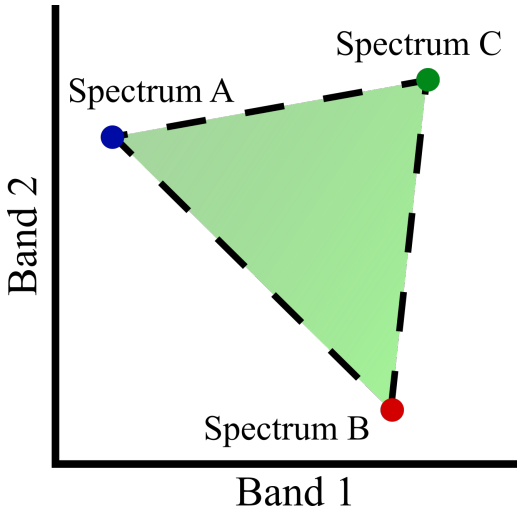
The 2D/3D scatter plots of PCs scores provide information about patters, trends, clusters and outlier identification. PCA has less discriminative efficiency since the method does not elaborate a model pattern or predictive approach.



(a)

4.1.2 Vertex component analysis (VCA)

Vertex component analysis. This an unmixing algorithm to find the endmembers of multidimensional datasets. VCA determines the vertexes of a geometrical simplex (extension of the triangle). VCA reduces the dimension of the datasets by principal componets or single value decomposition depending on the signal-noise ratio of the input data. The number of endmembers is defined by the user. After computing the endmembers, the concentrations are determined by algorithms that estimate concentrations, for example, non-negative least squares. A linear system is defined as eq. 7 shows, M is the endmember matrix, α is the abundance or concentration and *n* is the noise. VCA assumes the presence of pure endmember pixels and linear behavior Fig. 6a.



(a)

$$r = M\alpha + n \tag{7}$$

4.1.3 Multivariate curve resolution (MCR)

This algorithm finds the mixture elements by expressing the data by a bilinear model of pure components. Constrains and modern bilinear/multilinear models offer better perfomances. Nowadays MCR is a well established method for unmixing [**MCR**].

$$\mathbf{p} = \mathbf{C} \tag{8}$$

$$r = M\alpha + n \tag{9}$$

4.2 Clustering

Clustering discovers similarities in the dataset shape and join them into a converging number of clusters. Given a dataset A with n points x_i in d-dimensional space, and given the number of clusters k, the goal is to split the set of data into groups that hold similar characteristics, whereas points in different group are unrelated. The clustering is denoted $C = C1, C2, \ldots, Ck$. For each cluster, there is a representative point that summarizes the cluster, commonly called centroid μ_i.

$$\mathrm{u}_i = \frac{1}{ni} \sum_{x_j \in \varepsilon} X_j \tag{10}$$

Eq. 10 shows the centroids X dataset.

4.2.1 K-means analysis

K-means works generally well for general purposes. The algorithm tries to find n groups of equal variances, while minimizing a sum of distances. Given a clustering $C = C1, C2 \ldots, Ck$ and a function, which evaluates a sum of squared errors (SSE) (eq. 11).

$$SSE(C) = \sum_{i=1}^{k} \sum_{x_j \in C_i} |x_i - \mu_i|^2 \tag{11}$$

The final goal is to determine the clustering that minimizes $SSE(C)$ value (eq. 12):

$$C + minSSE(C) \tag{12}$$

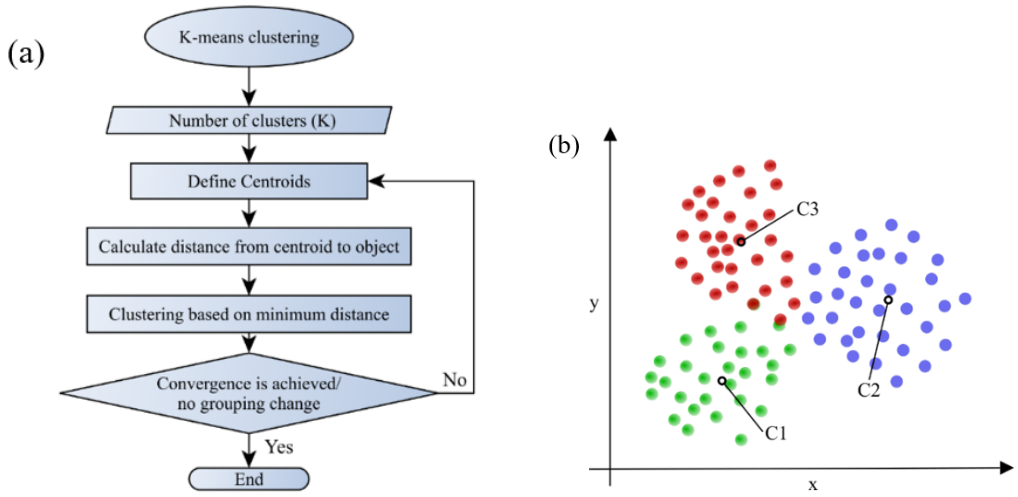
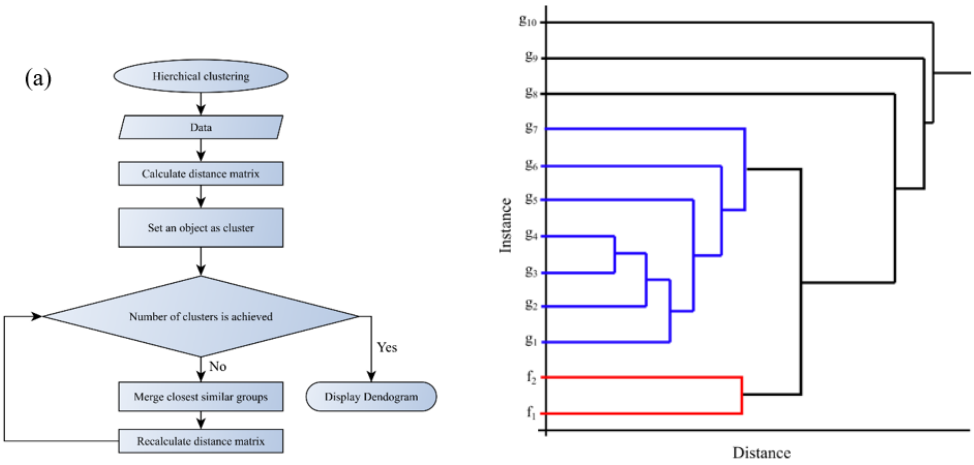


Figure 7: Kmeans, (a) algorithm and (b) clustering.

4.2.2 Hierarchical clustering analysis

The algorithm builds nested cluster by merging or splitting successively priori clusters. A dendrogram (or tree) displays the information presented by the clustering. The agglomerative clustering starts from bottom up (each observation is a cluster) and merges the clusters continuously. Generally, four linkage criteria are used: ward that minimizes the SSE within all clusters.

Complete linkage that minimizes the maximum distance between observations of pairs of clusters. Average linkage minimizes the average of the distances of all observations of pairs of clusters. And single linkage that minimizes the distance between the closest observations of pair of clusters [**hierarchical**]. For calculating the distances, several approaches come out. Euclidian, Minkowsky and Pearson distances.



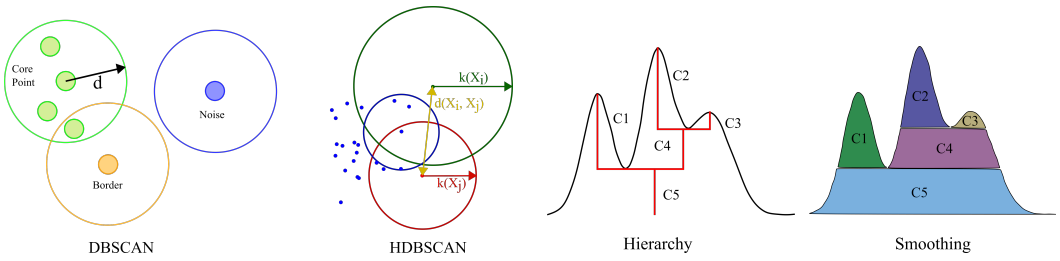
(a) (b)

Figure 8: HCA, (a) algorithm and (b) dendogram clustering.

4.2.3 Hierarchical density clustering with application with noise (HDBSCAN)

Hierarchial density-based spatial clustering applications with noise (HDBSCAN). It clusters points that keep similar density conditions in a defined number of neighbors *k*. Fig. 9a illustrates DBSCAN, which defines a fixed distance (d) and thus fixes density. The densiy is calculated by 1/d, thus the noisy points get low density and many points in d provide high densities. HDBSCAN uses a dynamic distance called mutual reacheability distance . finds the maximum distance from a certain number of neighbors at X_j, or X_i or distance from X_i and X_j. Fig. 9a shows HDBSCAN with calculated from X_i and neighbor point Xj. In this example, the largest distance is d. As the number of neighbors increases, more points are taken with large d values (noisy points). When *k* is set 1, the algorithm behaves like a hierarchical clustering with a single linkage (highly sensitive to noise). By computing the density, the noisy points obtain low density. The dynamic calculation of density allows to definition a hierarchy as Fig. 9b displays.

$$d_{\text{mreach}}(X_i, X_j) = \{max\{\kappa(X_i), \kappa(X_j), d(X_i, X_j)\} \quad X_i \neq X_j\} \tag{13}$$



(a) (b)

Figure 9: HDBSCAN, (a) representation and (b) distribution.

Advantages:

- HDBSCAN does not require to specify the number of expected clusters
- It can discover any shape of clusters.
- It defines a noise cluster and is robust to outliers.
- It requires two parametes, the number of neighbors, and the minmium size of clusters.

Disadvantages:

- It is not entirely deterministic.
- It overcomes the disadvantage of DBSCAN for finding clusters with different densities.

4.3 Supervised methods

4.3.1 PLS-LDA

5 Visualization

5.1 Spectra

5.2 Hyperspectral images

5.3 Depth profiling spectra

5.4 3D hyperspectral images

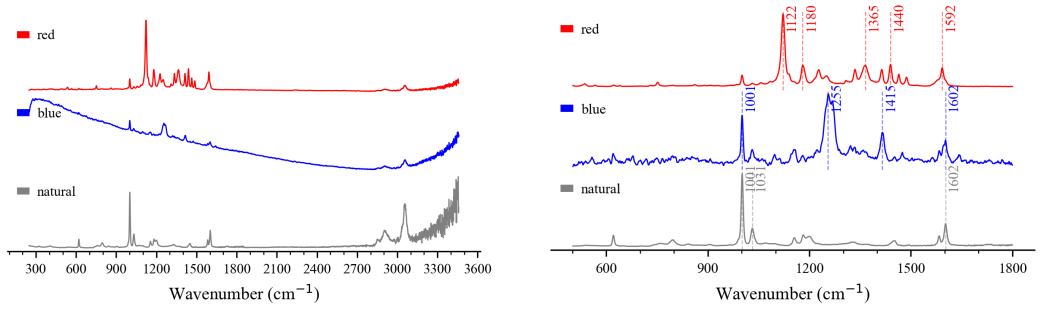
6 Examples

6.1 Plastics

```
from spectramap import spmap as sp
### Defining paths ###
blue_path = 'examples/plastics/blue'
red_path = 'examples/plastics/red'
natural_path = 'examples/plastics/natural'

### Creating and reading the files ###
## Declaring the first hyper object for red sample
red = sp.hyper_object('red')
## Reading the comma separated vector file
red.read_csv_xz(red_path)
## Setting red name to the whole data set spectra
red.set_label('red')
red.keep(500, 1800) ## Keep finger region
meanred = red.mean() ## Compute mean of all red spectra
## Same procedure for the blue sample
blue = sp.hyper_object('blue')
blue.read_csv_xz(blue_path)
blue.set_label('blue')
blue.keep(500, 1800)
blue.rubber() ## Baseline correction rubber band
meanblue = blue.mean()
## Same procedure for the natural sample
natural = sp.hyper_object('natural')
natural.read_csv_xz(natural_path)
```

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



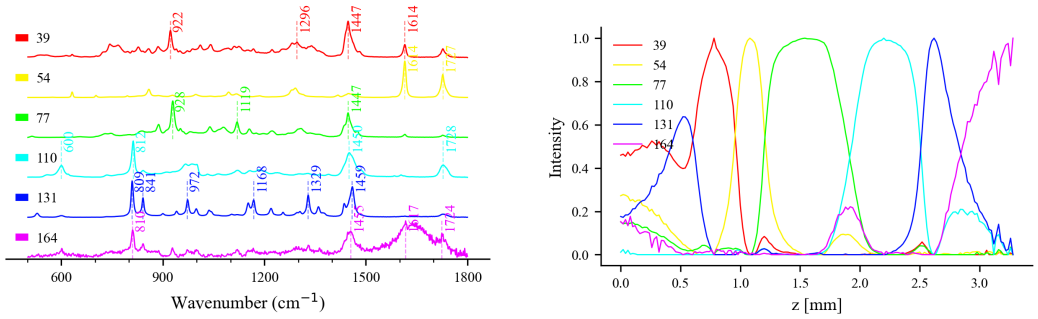
(a) (b)

Figure 10: SpMap processing, (a) Raw and (b) processed data

6.2 Depth profiling of multilayer of plastics

```
from spectramap import spmap as sp
### reading ###
path = 'examples/layers/layers'
stack = sp.hyper_object('layers') #creating the hyper object
stack.read_csv_xz(path) #reading the csv file Wavenumber (cm-1)
### preprocessing ###
stack.keep(500, 1800) #finger print selection
stack.rubber() #baseline correction
stack.vector() #vector normalization
### processing ###
endmember = stack.vca(6) # vertex component analysis
# visualization of spectra and strong peaks(Fig.5.6(a))
endmember.show_stack(0.3, 0, 'auto')
# concentration estimation by NNLS
abundance = stack.abundance(endmember, 'NNLS')
 # set resolution of 20 µm for the profile
abundance.set_resolution(0.02)
```

```
# plot profile (Fig.5.6(b))
abundance.show_profile('auto')
```

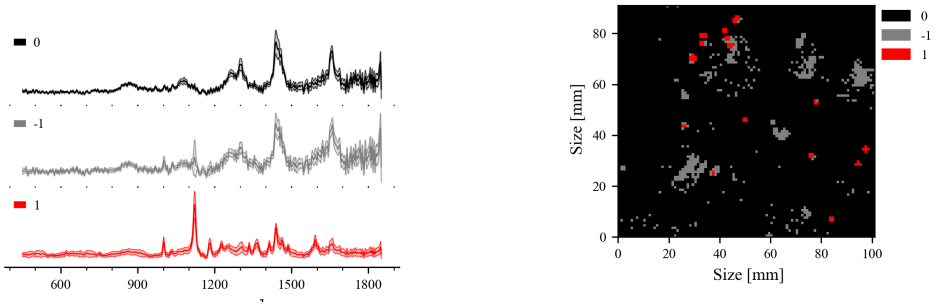


(a) (b)

Figure 11: Example 2, (a) plastics and (b) depth profile

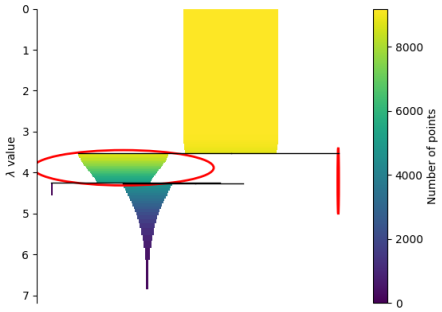
6.3 Microplastics in tissue

```
#For this example is required to enable hdbscan
#in spmap (uncomment the lines)
from spectramap import spmap
micro = spmap.hyper_object("data") #hyperobject
micro.read_csv_xz("microplastics_tissue") #reading 2D data
#Hierchical density-based clustering
micro.hdbscan(2, 15)
#mean spectra of the clusters
micro.show_stack(0,0, 'auto')
#2D map of the clusters
micro.show_map('auto', None, 1)
```



(a)

(b)



(c)

Figure 12: Example 3, (a) spectra, (b) clustered image and (c) hdbscan dendogram

6.4 3D hyperspectral image

```
from spectramap import spmap as sp
import pandas as pd

### reading ###
path = 'examples/3D/cube'
cube = sp.hyper_object('cube') #creating the hyper object

pre_result = pd.read_table(path+'.csv.xz', sep=',')
#reading the 3d csv file and placing theresolutions xy = 60 µm and z = 70 µm
cube.read_csv_3d_xz(path)
### 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')
# concentration estimation by NNLS
abundance = cube.abundance(vca, 'NNLS')
# 3d plot of all clusters (Fig. 5.8(b-c))
aux = abundance.show_intensity_volume(0.5)
```

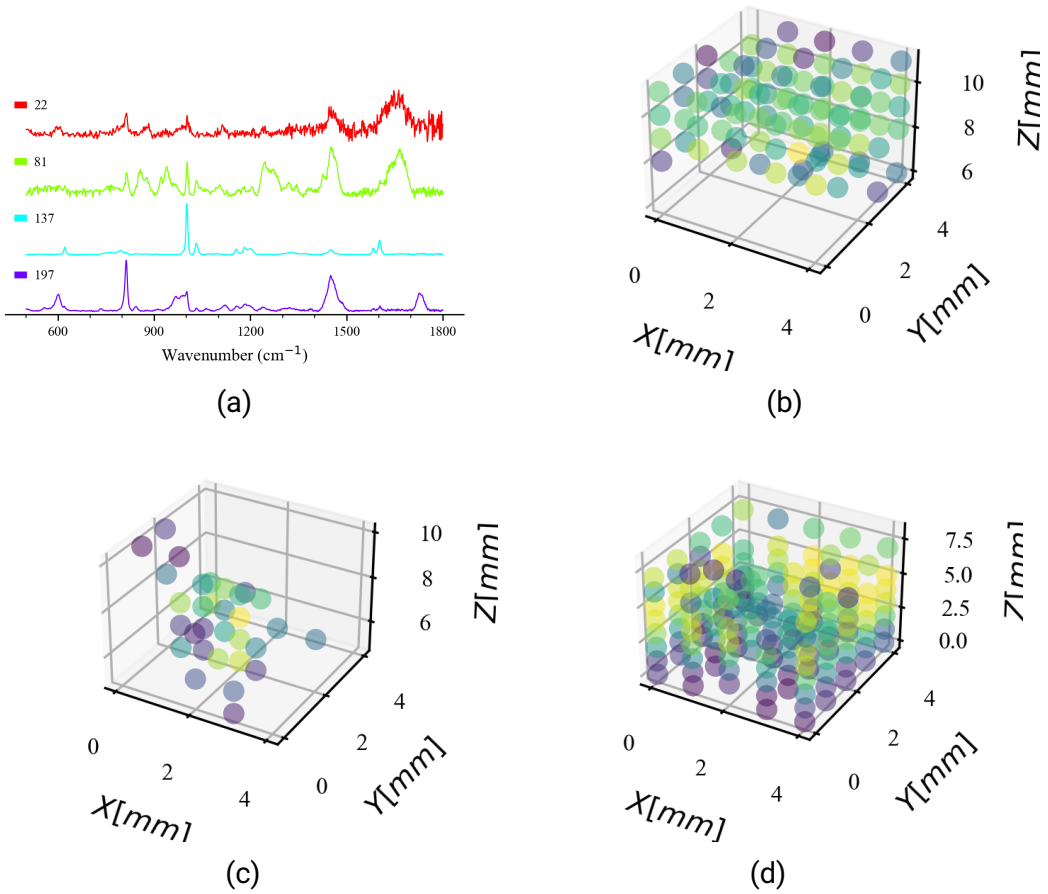


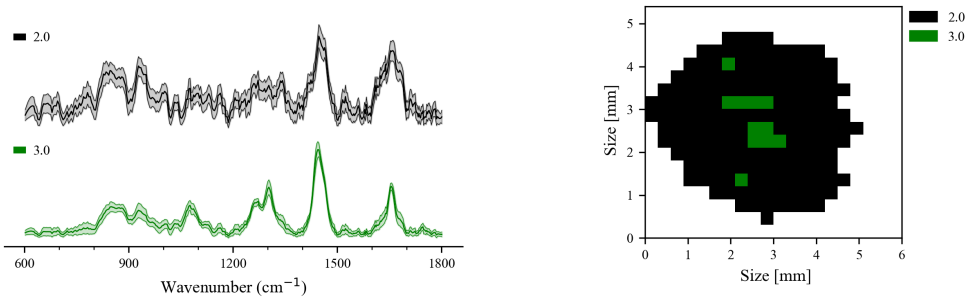
Figure 13: Example 4, (a) VCA results, (b) upper layer (tissue-green color), (c) microplastic (cian) and (d) lower layer (PMMA - purple)

6.5 Infrarred Image

6.6 Bladder

```
from spectramap import spmap as sp
bladder = sp.hyper_object("bladder")
bladder.read_csv_xz("bladder")
```

```
bladder.set_resolution(0.3) ## 300 μm step size resolution
bladder.vector() # vector normalization
original = bladder.get_data() ## get data
### K-means clustering
bladder.kmeans(3) #K-means clustering: 3 components
bladder.remove_label([1])
colors = bladder.show_map(['black', 'green'], None, 1) #
bladder.show_stack(0, 0, colors) #show stack of the clusters (Fig. 5.5(c))
```



(a) (b)

Figure 14: Example 5, (a) spectra of bladder and (b) hyperspectral image.

6.7 Unmixing and clustering of Bladder Hyperspectral Images

```
from spectramap import spmap as sp
#%% reading csv.xz files
bladder = sp.hyper_object("bladder") ## the library
bladder.read_csv_xz("long_bladder")
#%% first sight of the spectra
peaks = bladder.show2(True, 0.2, "r") # basic plotting
#%% processing data
bladder.rubber() # rubber band baseline (basic)
bladder.gol(11, 3, 0) #savitzky golay filter
## remove lipids
bladder.remove_label(['Lipid+Protein', 'Protein+Lipid',
'Tumor+Lipid'])
## remove nonclear data
bladder.rename_label(["Tumor+Fibrosis+Protein", "Necrosis+Tumor",
"Tumor+Necrosis"], ["Tumor", "Necrosis", "Necrosis"])
bladder.rename_label(["Tumor", "Necrosis", "Protein", "Lipid"],
["T", "N", "P", "L"]) # rename labels
bladder.show_spectra(0, 0, 'auto')
## principal component analysis
```

```
scores, loadings = bladder.pca(3, False)
## show stacked plot
loadings.show_stack(0.2,1,"auto")
## show scatter plot
scores.show_scatter(bladder.get_label(), 18, "auto")
# hierarchical clustering analysis
bladder.hca("euclidean", "ward", 0.5, 12)
scores.show_scatter(bladder.get_label(), 18, "auto")
bladder.show_spectra(0,0,"auto")
```

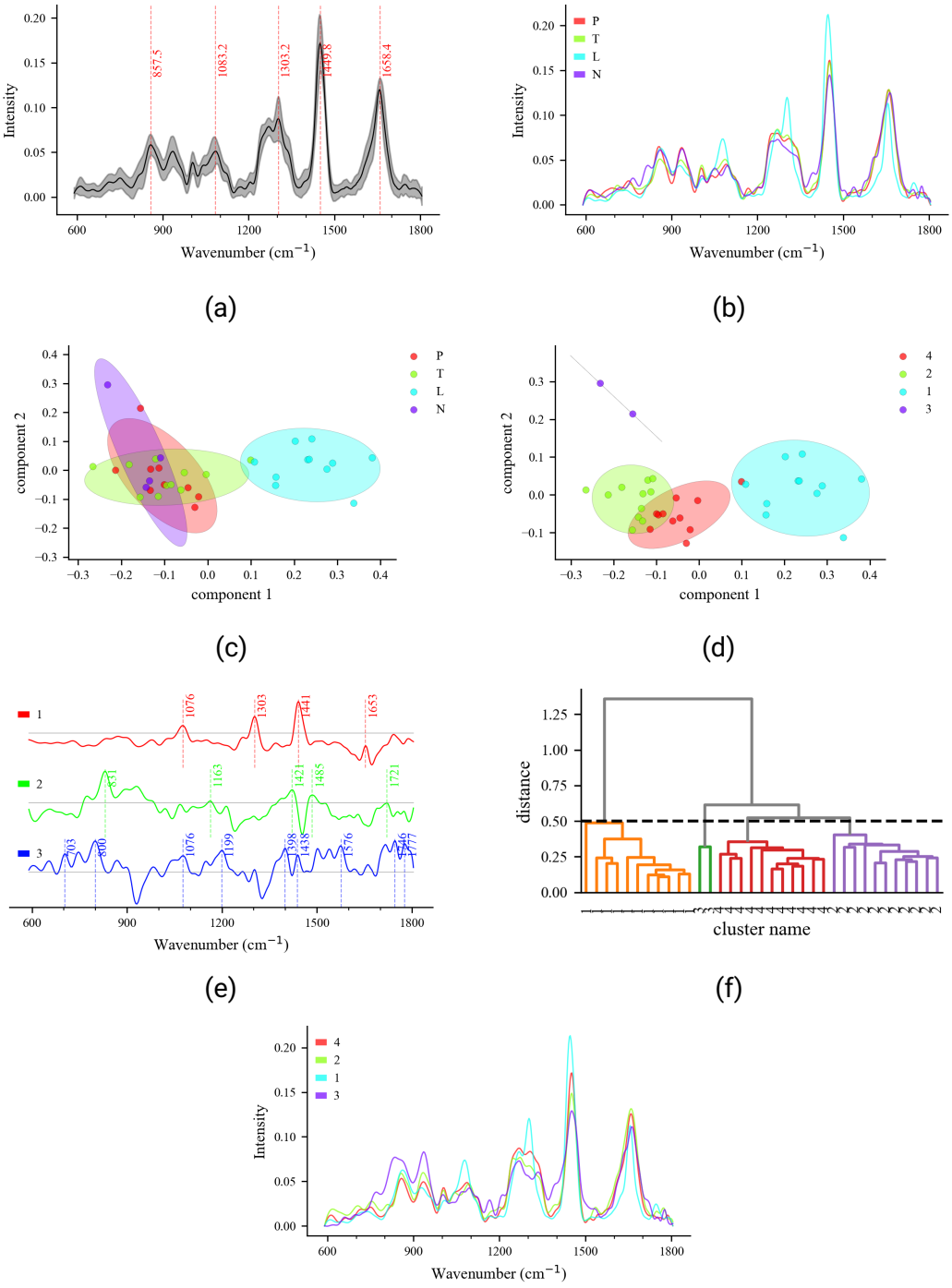
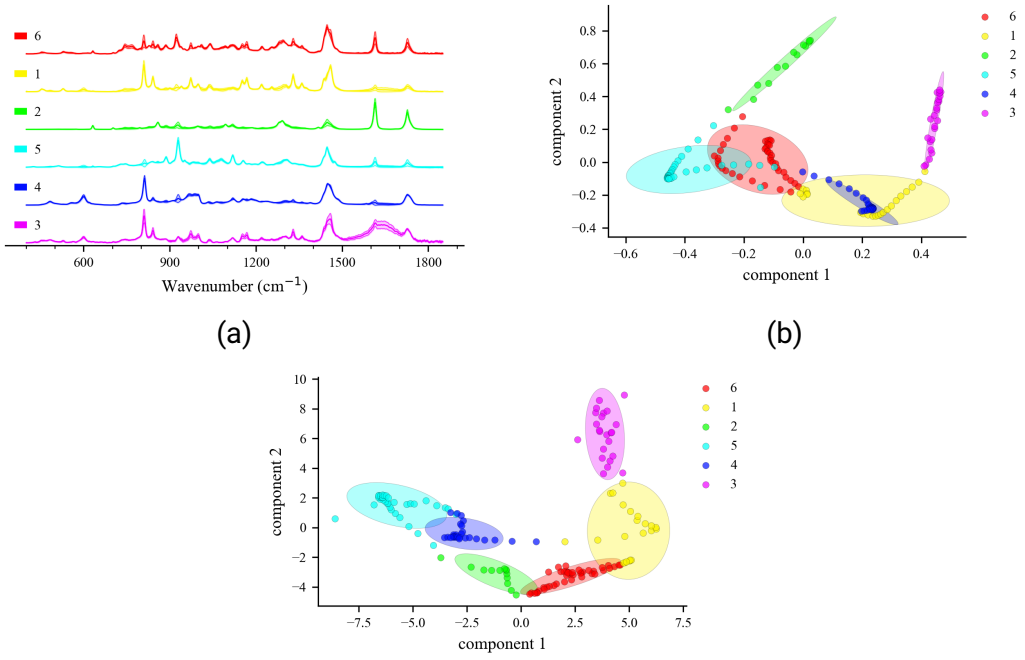


Figure 15: Example 6, (a) raw average, (b) clean data, (c) PCA scatter of scores, (d) PCA scatter using HCA clustering, (e) PCA components, (f) HCA dendogram and (g) cluster components.

6.8 Unmixing PCA and PLS-LDA of plastics

```
from spectramap import spmap as sp
plastics = sp.hyper_object('plastics')
plastics.read_csv_xz('layers')
plastics.show(True)
# keeping finger print region
plastics.keep(400, 1850)
plastics.gaussian(2) # appliying gaussian filter
plastics.rubber() # rubber baseline correction
plastics.vector()
plastics.kmeans(6) # kmeans clustering example for main_label
main_label = plastics.get_label() # saving the main_label
main_label.name = "main_label" # renaming the title of the label
plastics.show_stack(0,0, "auto") # showing the 6 components
# 3 components pca
scores_pca, loadings_pca = plastics.pca(3, False)
# showing scatter with sublabel
scores_pca.show_scatter(main_label, 15, "auto")
# 3 components pls-lda  and 70% training data
scores_pls, loadings_pls = plastics.plslda(3, 1)
# showing scatter with sublevel
scores_pls.show_scatter(main_label, 15, "auto")
```



(a) (b)

(c)

Figure 16: Example 8, (a) plastics, (b) PCA scores and (c) PLSLDA scores

Bibliography

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

[2] J. Healy L. McInnes and S. Astels. “hdbscan: Hierarchical density based clustering In: Journal of Open Source Software”. In: *The Open Journal* 2.4 (2017).

[3] J. M. P. Nascimento and J. M. B. Dias. “Vertex component analysis: A fast algorithm to unmix hyperspectral data”. In: *IEEE Transactions on Geoscience and Remote Sensing* 43 (2005), pp. 898–910. DOI: 10.1109/TGRS.2005.844293.

[4] R.W. Schafer. “What is a savitzky-golay filter”. In: *IEEE Signal Process* 28 (2011), pp. 111–117.

[5] S. Chen Z. M. Zhang and Y. Z. Liang. “Baseline correction using adaptive iteratively reweighted penalized least squares”. In: *Analyst* 5 (2010), pp. 1138–1146. DOI: 10.1039/b922045c.