Robust Automatic Identification of Microplastics in Environmental Samples using FTIR Microscopy – Supporting Information

Gerrit Renner,[†,‡] *Philipp Sauerbier,*[¶] *Torsten C. Schmidt,*[‡] *Jürgen Schram*[*,†]

† *Instrumental Analytical and Environmental Chemistry, Faculty of Chemistry, Niederrhein University of Applied Sciences, Frankenring 20, D-47798 Krefeld, Germany*

‡ *Instrumental Analytical Chemistry and Centre for Water and Environmental Research (ZWU), University of Duisburg-Essen, Universitätsstr. 5, D-45141 Essen, Germany*

¶ *Wood Biology and Wood Products, Faculty of Forest Sciences, University of Goettingen, Büsgenweg 4, D-37077 Göttingen, Germany*

E–mail: juergen.schram@hs-niederrhein.de
Phone: +49 (0)179 1043284

Contents

In–House Reference Spectra Library

Table S 1: Overview of the In–House Reference Database. All Materials were measured with FTIR ATR and FTIR microscopy. For identification, ATR data were compared to the ATR spectra library and µFTIR data were compared to the µFTIR spectra library.

Polymer	Label	Shape	Diameter	Source
Polyethylene	LDPE	granules	5 mm	*Carat GmbH*
Polypropylene	PP	granules	5 mm	*Carat GmbH*
Polyamide	PA-6	granules	5 mm	*Carat GmbH*
Polyvinyl chloride	PVC	powder	0.1 mm	*Solvay*
Polycarbonate	PC	granules	5 mm	*Covestro AG*
Polyurethane	PU	flakes	20 mm	*Carl Bernh. Hoffmann GmbH & Co. KG*
Polystyrene	PS	film	0.1 mm	*Thermo Fisher Scientific Inc*
Polyethylene terephthalate	PET	granules	5 mm	*Carat GmbH*
Polymethyl methacrylate	PMMA	powder	0.5 mm	acrylic glass sheet (milled)
Chitin	Chitin	powder	0.5 mm	shrimps (shells milled)
Polybutadiene	BR	fragments	0.5 mm	car tires (milled)
Polylactide	PLA	powder	0.5 mm	3D printer filament (milled)
Polyoxymethylene	POM	powder	0.5 mm	conical joint clips (milled), *Carl Roth GmbH + Co. KG*
Polytetrafluoroethylene	PTFE	flakes	1 mm	thread sealing tape
Celluloid	Celluloid	flakes	1 mm	photography film
Cellulose Acetate	CA	fibers	1 mm	cigarette filters
Wood-plastic composite ($PP_{30\%wt.}$)	WPC	granules	5 mm	*Wood Biology & Products, University of Goettingen*
Parafilm	Paraffin wax / PE	flakes	1 mm	*Carl Roth GmbH + Co. KG*
Natural Rubber	Rubber	fragments	0.5 mm	natural rubber (milled)
Neoprene	Neoprene	flakes	1 mm	laboratory gloves, *Carl Roth GmbH + Co. KG*

Examined Library Searching Algorithms

Table S 2: Overview of the examined algorithms for hit quality index calculation.*

Algorithm	Formula
Euclidean Distance	$\left[1+\frac{\|\mathbf{s}-\mathbf{r}\|^2}{\|\mathbf{r}\|^2}\right]^{-1}$
Pearson r^2	$\frac{\mathbf{s}\cdot\mathbf{r}}{\|\mathbf{s}\|\,\|\mathbf{r}\|}$
Pearson r^2 *1st Derivative*	$\frac{\partial\mathbf{s}\cdot\partial\mathbf{r}}{\|\partial\mathbf{s}\|\,\|\partial\mathbf{r}\|}$
Pearson r^2 *2nd Derivative*	$\frac{\partial^2\mathbf{s}\cdot\partial^2\mathbf{r}}{\|\partial^2\mathbf{s}\|\,\|\partial^2\mathbf{r}\|}$
µIDENT	$1-\left(\frac{\sum d_i\cdot w_i}{\sum w_i}\right)^W$

* **s** represents sample data, **r** represents reference data, d represents partial distance between two vibrational band area ratios, w describes a partial weighting based on statistical significance and W describes overall weighting that considers number of reference signals found in sample data.

Spectra of Case Study 1

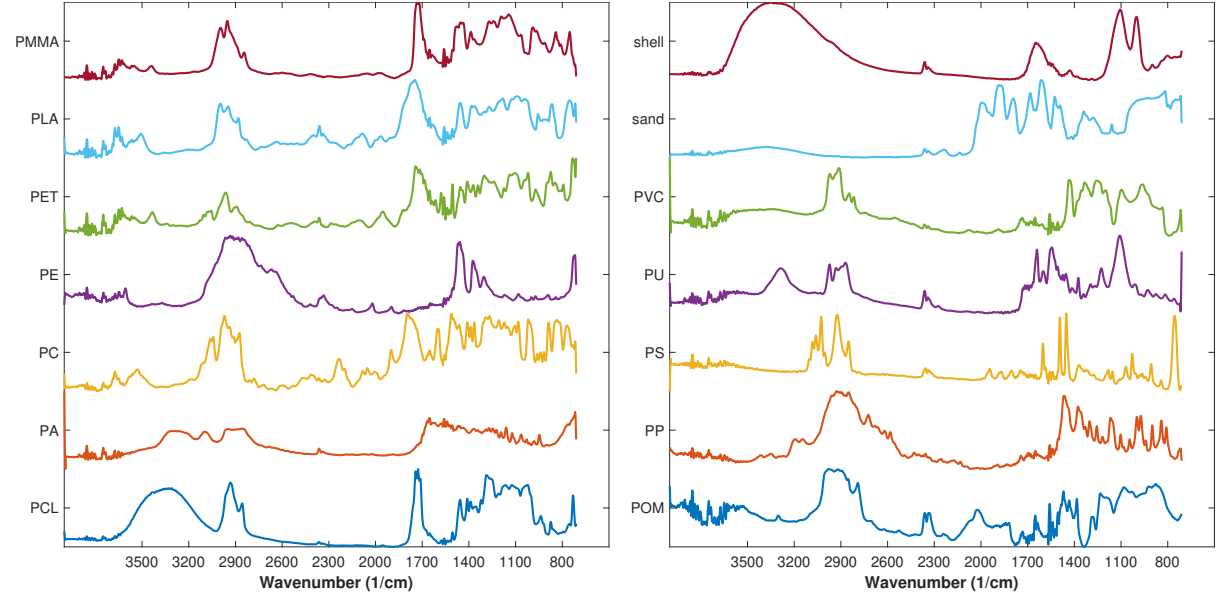


Figure S 1: Infrared microscope absorbance spectra of selected reference materials. PCL: polycaprolactone, PA: polyamide, PC: polycarbonate, PE: polyethylene, PET: polyethylene terephthalate, PLA: polylactide, PMMA: poly(methyl methacrylate), POM: polyoxymethylene, PP: polypropylene, PS: polystyrene, PU: polyurethane, PVC: polyvinyl chloride, shell limestone and sea sand particles

Spectra of Case Study 2

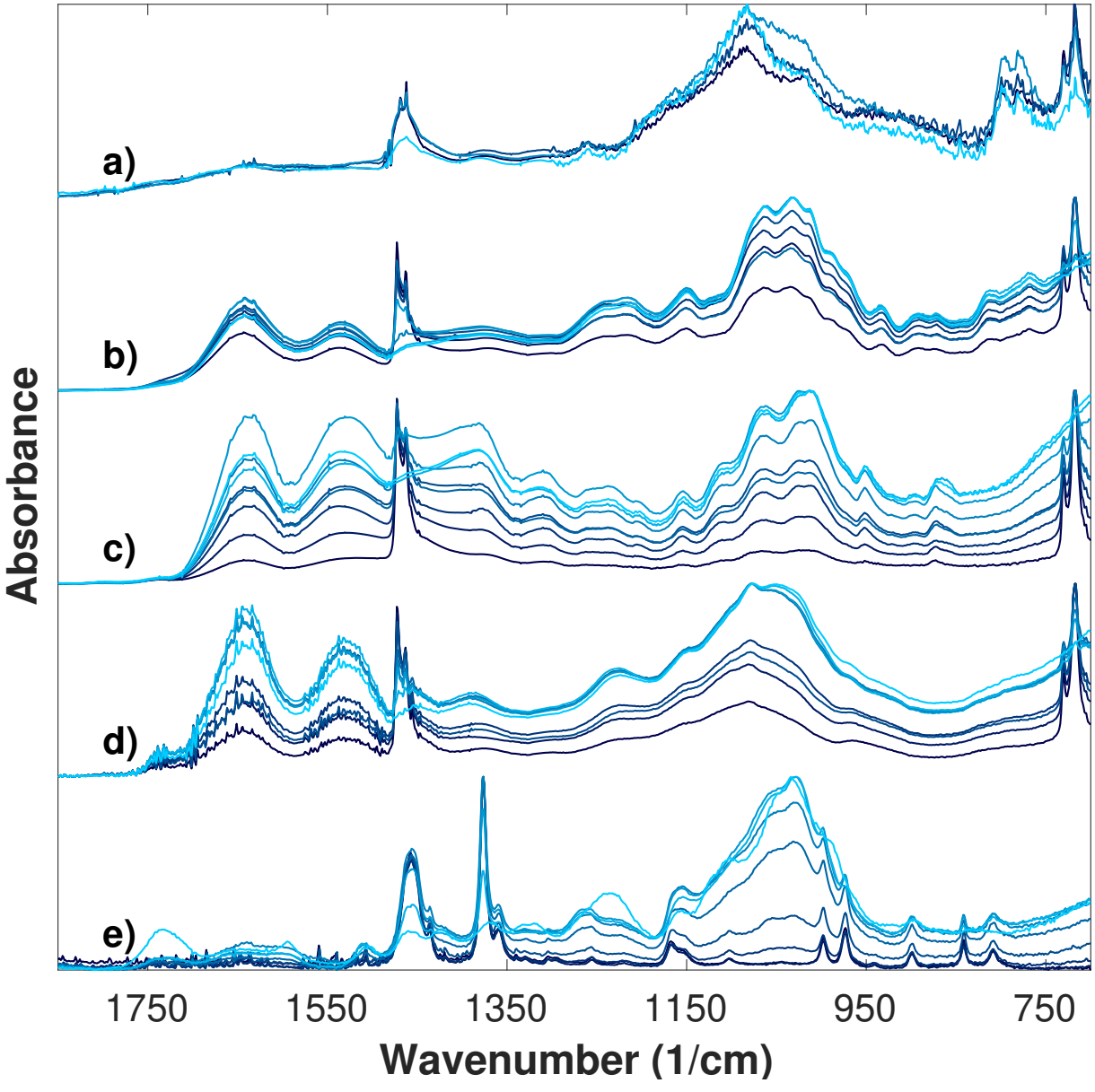


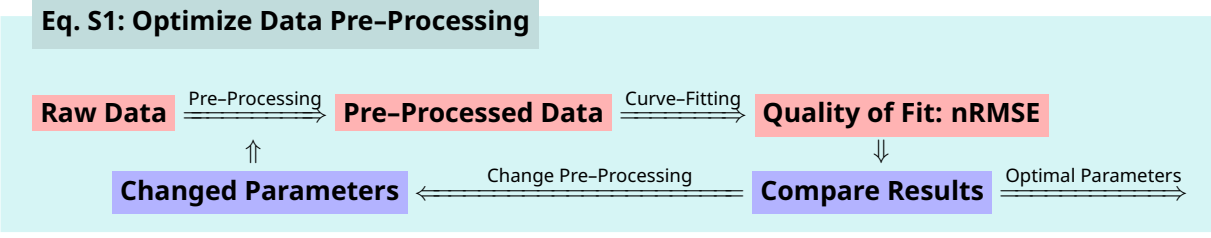
Figure S 2: FTIR ATR spectra of microplastics in presence of interfering matrices. Every mixture ranges between low (dark blue) and high (light blue) matrix content. a) PE + sand (Matrix: 20, 40, 60, 80 $\%_{wt.}$), b) PE + algae (*Porphyra*) (Matrix: 17, 20, 25, 33, 50, 67, 75, 80, 83 $\%_{wt.}$), c) PE + chitosan (Matrix: 10, 20, 30, 40, 50, 60, 70, 80, 90 $\%_{wt.}$), d) PE + algae (*Chlorophyta*) (Matrix: 10, 20, 30, 40, 50, 60, 70, 80, 90 $\%_{wt.}$), e) PP + wood (spruce) (Matrix: 0, 10, 20, 30, 40, 50, 60, 70, 100 $\%_{wt.}$).

Wood–Polymer–Composite Manufacturing
Particles with a grain size of 70–150 µmwere used for wood powder softwood (*Arbocel C 100; J. Rettenmaier & Söhne GmbH und Co.KG, Rosenberg, Germany*). Polypropylene (PP) was used as a polymer matrix (*SABIC 575P, Riyadh, Saudi Arabia*). Maleic anhydride-grafted polypropylene (MAPP) as coupling agent was also added to the composites for an application-oriented and realistic investigation. The amount of coupling agent was fixed to 3 $\%_{wt}$ of the PP.

The WPC was compounded in a *Leistritz MICRO27GL/GG40D* co-rotating twin-screw extruder (*Leistritz Extrusionstechnik GmbH, Germany*) with gravimetric feeders (*Brabender GmbH & Co. KG, Germany*) and a hot-cut pelletizer. The granulates were produced with a mould temperature of about 140 °C. A total of eight WPC formulations were produced, with the wood content ranging from 0–70 $\%_{wt}$ in 10 $\%_{wt}$ increments.

Optimization of Data Pre–Processing Steps

The presented new modified microplastics identification algorithm is based on diverse data pre–processing steps. Thereby, the optimizations of these steps were performed using 200 FTIR microscopy spectra of microplastics reference particles made of polypropylene, polyamide and polylactide. The primary objective is to describe every FTIR spectrum by an individual cumulative pseudo *Voigt* function as precise as possible. To that end, a suitable data pre–processing leads to highest similarity values between pre–processed measurement data and fitted data. In other words, the error (normalized root–mean–square error, nRMSE) should become as small as possible. The optimization process is illustrated in Eq. S1



The data pre–processing optimizations were performed for two independent steps: *baseline correction* and *data smoothing*. Within both steps there were different algorithms tested and their resulting nRMSE values were compared with each other. However, to find optimal parameters for every individual pre–processing algorithm, *Downhill–Simplex–Optimizations* according to *Nelder and Mead* were performed focusing on the minimization of nRMSE. As input, reference spectra of PP, PA and PLA were treated by the individual algorithms with discrete parameter settings, and as output, the means of nRMSE were recorded, while the latter was used as the optimization target. Baseline correction was optimized initially and the best method was used for further optimizations of data smoothing. All examined algorithms with their best target values and optimized parameters can be seen in Table S 3.

Table S 3: Experimental Design to Optimize Baseline Correction and Data Smoothing of Infrared Spectra of Microplastics.

Pre–Processing Algorithm	**Polypropylene** nRMSE (%)	**Polyamide** nRMSE (%)	**Polylactide** nRMSE (%)	**Average** nRMSE (%)
Baseline Correction				
Rolling circle filter	12.34	11.85	13.52	12.57
Adaptive iteratively reweighted penalized least squares smoothing (airPLS)	9.05	9.44	9.49	9.32
Wavelet transformation	10.22	11.55	12.03	11.27
Baseline Correction (airPLS) + Data Smoothing				
Rolling circle filter	5.93	6.00	6.01	5.98
Adaptive iteratively reweighted penalized least squares smoothing	8.88	8.58	9.13	8.86
Wavelet transformation	4.78	5.12	4.98	4.96
Moving Average	6.84	7.08	6.65	6.86
Locally weighted scatterplot smoothing	5.21	5.55	4.23	5.00
Savitzky–Golay filter (static smoothing span)	3.14	6.12	3.84	4.37
Savitzky–Golay filter (variable smoothing span)	2.67	5.20	3.16	3.68

Smoothing with Variable Span

Noise levels of infrared spectra are rarely constant but often variable. One origin of this phenomenon is based on the variances of water related signals. Actually, water related infrared signals are no noise but they behave very similar and disturb the measured data significantly even after performing water and atmosphere compensation, which can be seen in Figure S 3.

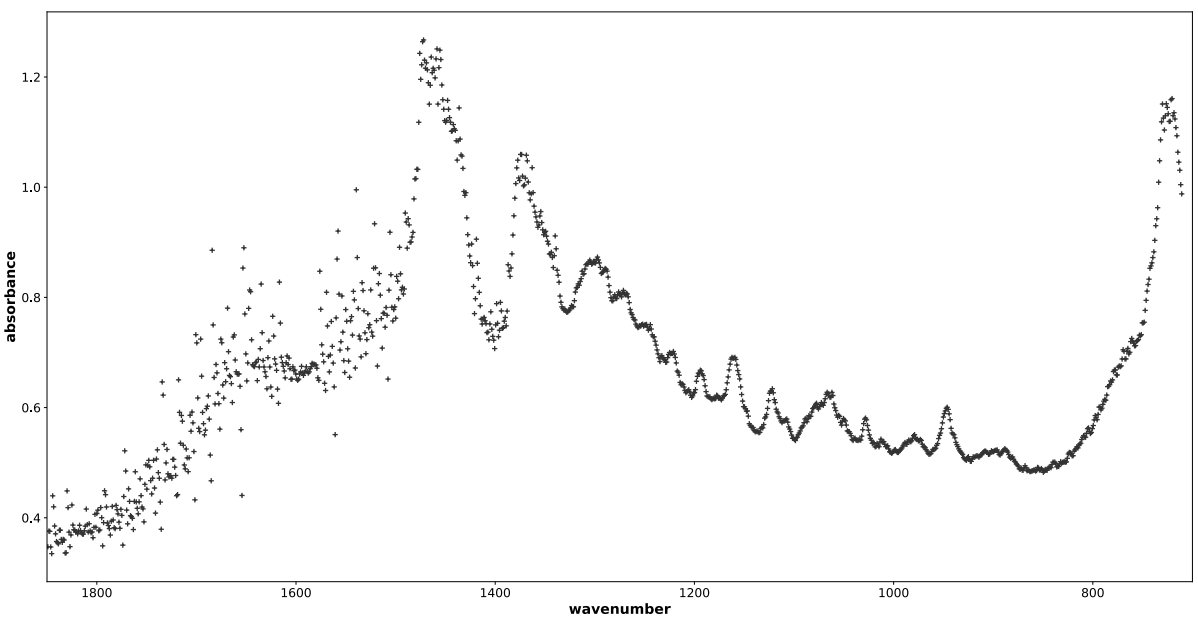


Figure S 3: µFTIR spectrum of polyethylene microplastics after water and atmosphere compensation. The given spectrum still contains significant noisy disturbances, and therefore, it has to be smoothed in further spectrum treatment.

In the shown scenario, a constant smoothing span is not suitable, as the smoothed spectrum will be ”under–” or ”over–smoothed”, respectively, due to the variable behaviour of the overlaying noise. To that end, a smoothing method was slightly modified to work with a variable smoothing span, which can be seen in Figure S 4.

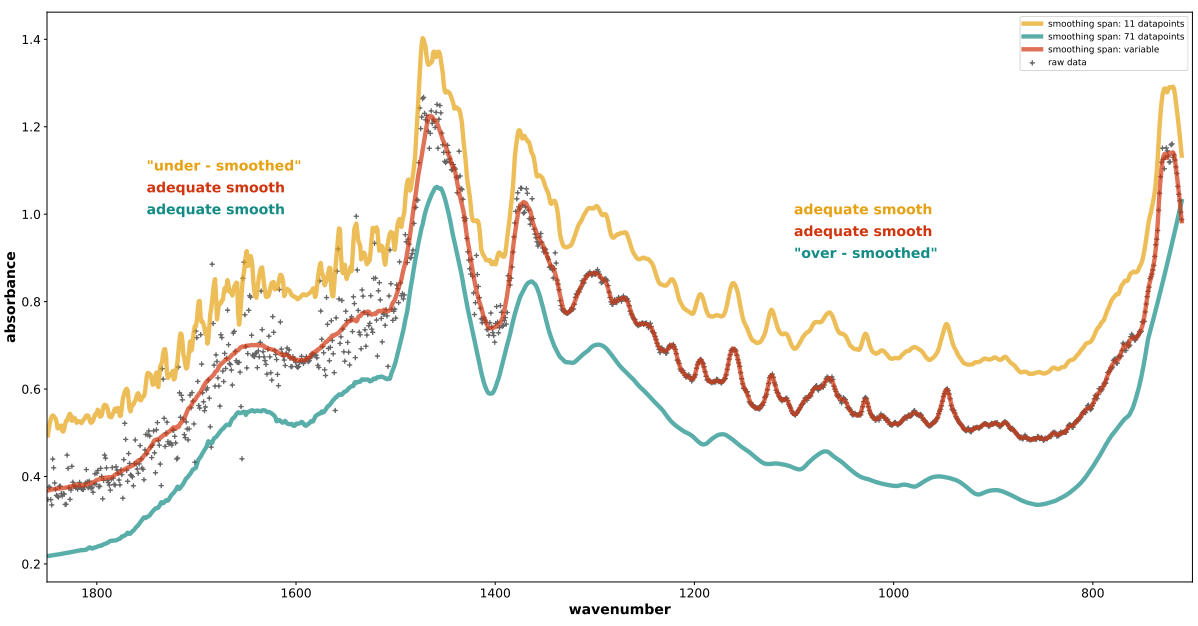


Figure S 4: µFTIR spectrum of polyethylene microplastics after water and atmosphere compensation (+). Additionally, three different smoothing spans were tested. For **yellow**, a small span of 11 data points were used to perform a *Savitzky–Golay* smoothing. In lower wavenumbers, this approach is suitable but in higher ranges the smoothed data still contain significant level of noise. For **green**, a broad span of 71 data points were used to perform a *Savitzky–Golay* smoothing. In higher wavenumbers, this approach is suitable but in lower ranges the smoothed data show ”over–smoothing”. For **red**, a variable span was used to perform a modified *Savitzky–Golay* smoothing. The smoothing concept is suitable for the complete range of the infrared spectrum.

To modify the used *Savitzky–Golay* smoothing algorithm, the spectrum is separated into multiple overlapping sections of 360 data points. The overlapping range is set to 60 data points, which means that every data point is smoothed 5 times with individually optimised smoothing spans, which can be seen in Figure S 5.

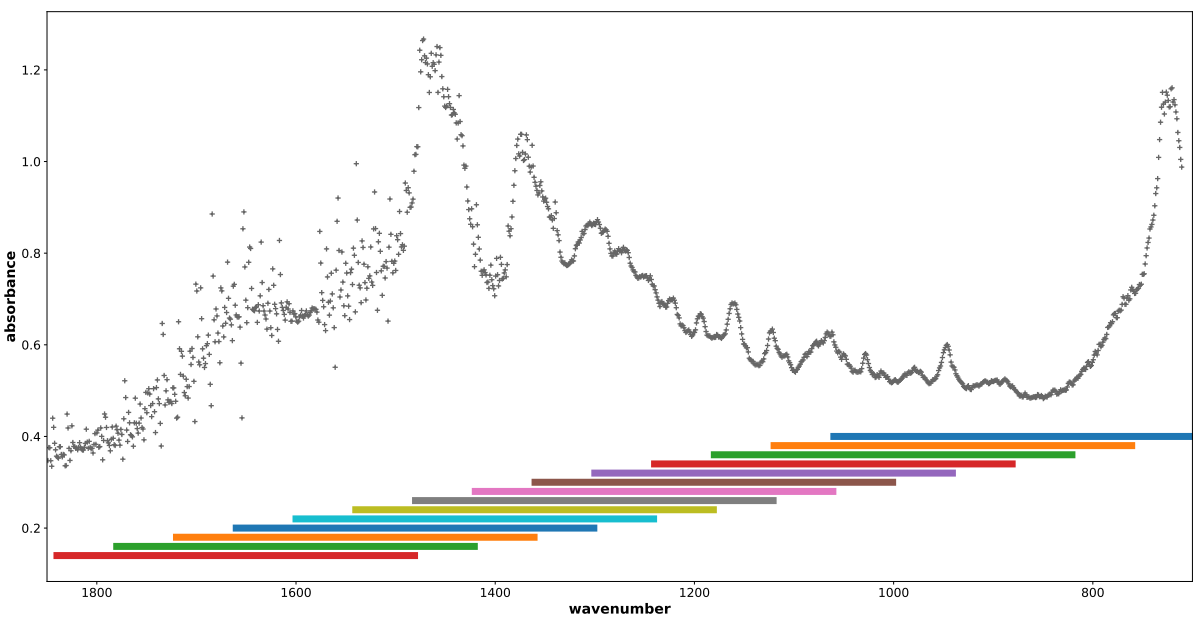


Figure S 5: µFTIR spectrum of polyethylene microplastics after water and atmosphere compensation. The coloured bars indicate the overlapping subdivided sections.

In a second step, each separated data range is smoothed individually by *Savitzky–Golay* smoothing. However, to create a suitable and practical algorithm, finding the optimal smoothing span has to be automated, which requires a target value that describes the quality of the iteratively performed smooth. Mathematically, it can be defined Eq. S2.

Eq. S2

Step 1: Define a function that connects *smoothing span* and *smoothing quality*.

$$f : \underbrace{\textit{smoothing span}}_{w} \rightarrow \underbrace{\textit{smoothing quality}}_{z}$$

Step 2: Optimise *smoothing quality* (z) by changing *smoothing span* (w).

Describing the quality of a performed smoothing process is a complex task. One suitable way is to apply *Fourier transformations* on smoothed and raw data. With this approach, two information can be obtained – individual *frequencies* that are part of the transformed data and their *amplitude*. If data is noisy, many different frequencies with highly variable amplitudes will appear in the transformed data set. If the data is smoothed well, significantly less frequencies with smaller and more homogeneous amplitudes can be observed in the transformed data set. This phenomenon can be used to describe a suitable target value by looking at the distribution (histogram) of the amplitudes and estimating the width of this distribution curve, which can be seen in Figure S 6.

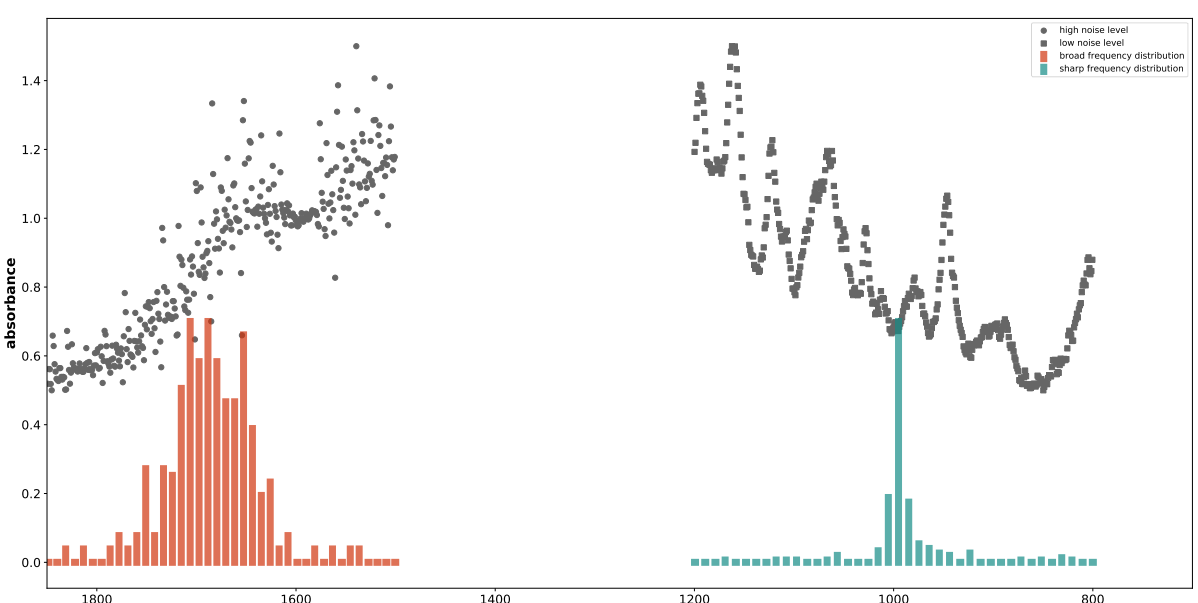


Figure S 6: Two different sections of µFTIR spectrum of polyethylene microplastics after water and atmosphere compensation. The coloured bars describe the histogram of the amplitudes after individual *Fourier transformations*.

Changing iteratively the smoothing span, will also change the width of the amplitude distribution peak. If a defined threshold criterion is reached, a suitable smoothing span for the analysed section can be extracted which can be seen in Figure S 7.

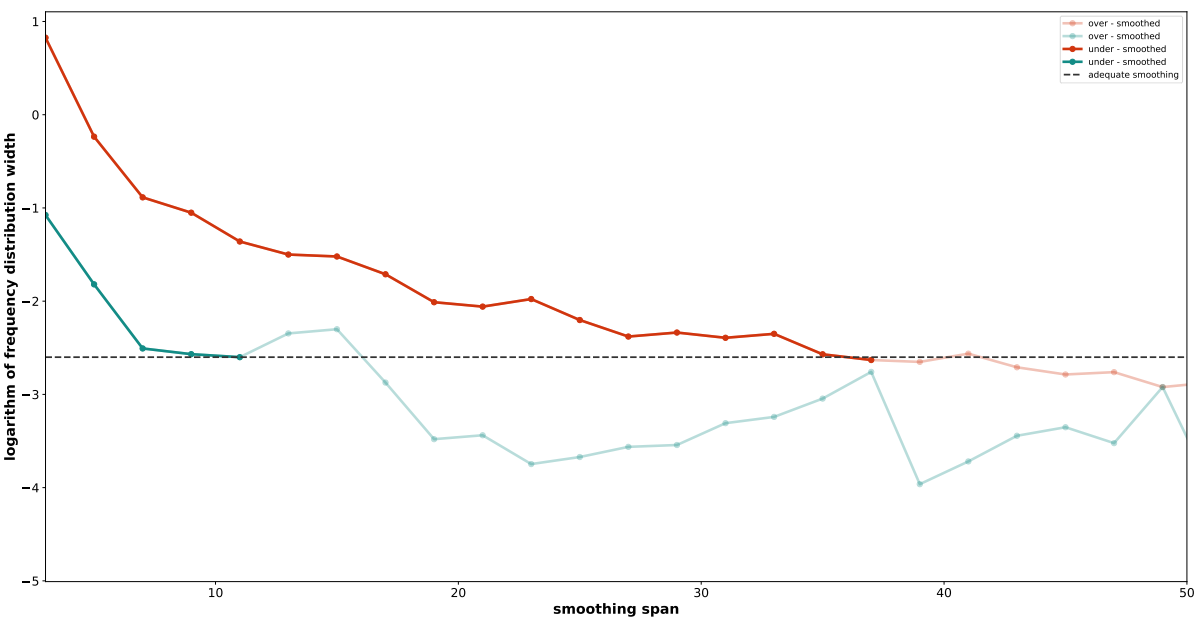


Figure S 7: Target values of two different sections within the FTIR fingerprint of polyethylene microplastics. Smoothing span is optimal if the empirically defined threshold is reached. While the red curve shows the smoothing span optimisation of a noisy data, which leads to relatively large smoothing spans (≈ 40), the optimal smoothing span for section with less noise interferences is significantly lower (≈ 10).

After individual smoothing span optimisation, all smoothed sections were re-concatenated. As every data point is described by multiple smoothings caused by section overlapping, median for each wavenumber is extracted. As a result, the optimised smoothed infrared spectrum with a variable smoothing span can be obtained.

An example of how the comparison algorithm works?
At the end of the presented data pre–processing method the incoming raw FTIR spectra were transformed into compact lists that contain vibrational band positions, areas and weighting factors. Every sample or reference is connected to an individual list. For *Hit Quality Index* calculation the similarity of those lists has to be estimated, and therefore, a special algorithm was developed[1], which will be explained based on a graphical example in the following. However, a more detailed description can be found in the former µIDENT study[1].
The main idea of the comparison algorithm is not to compare the vibrational band positions and areas directly but first to create a highly characteristic pattern of all unique vibrational band pairs and calculate their area ratios, which can be seen in Fig. 8.

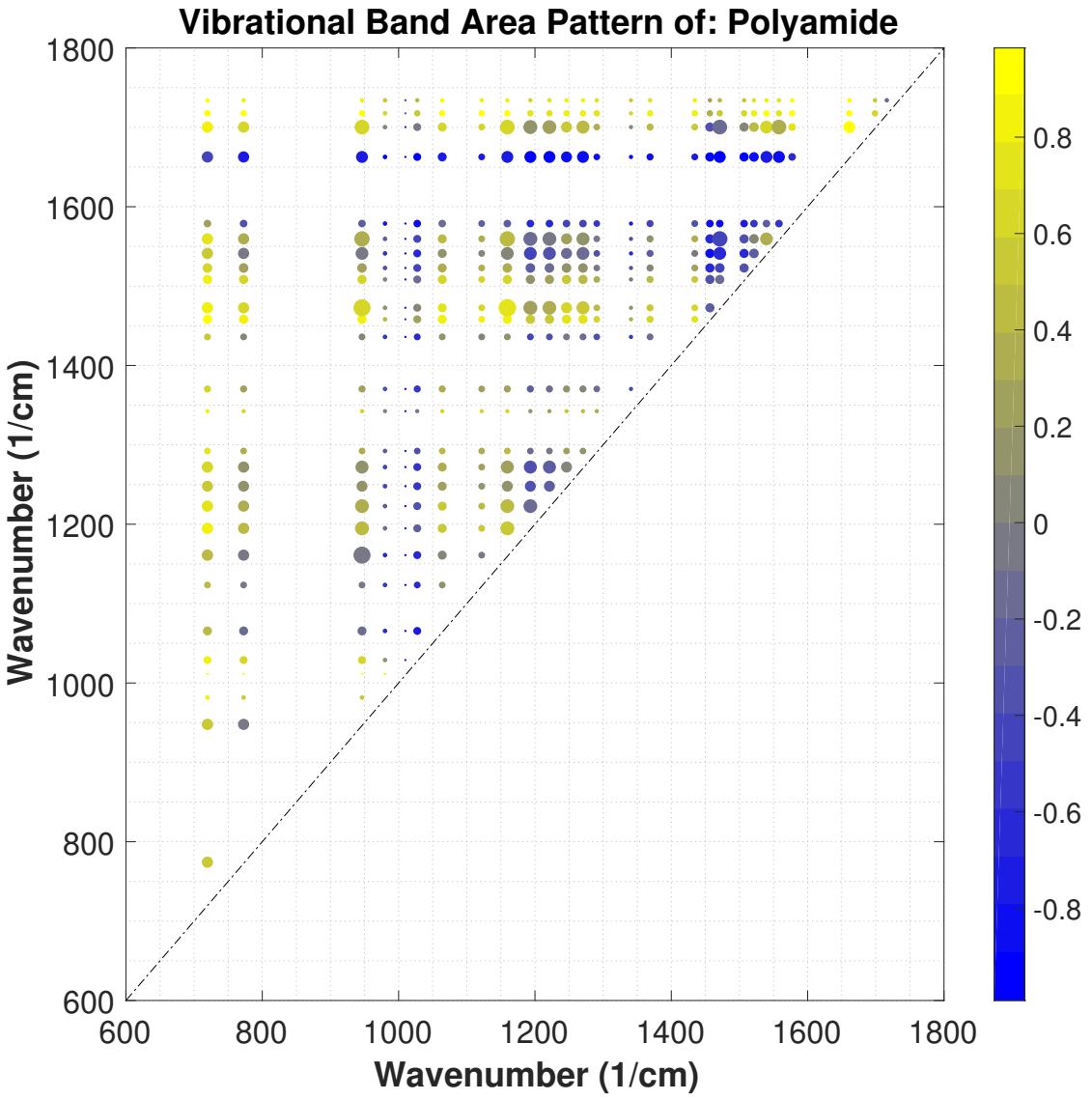


Figure S 8: Vibrational Band Area Pattern of Polyamide. The Scatter plot shows multiple colored different sized circles. Every circle represents an individual vibrational band pair within the FTIR spectrum of polyamide. The x/y coordinates are connected to the band pair positions. The circle color is the virtual third dimension of this diagram and represents the normalized area ratio of the band pair, which was calculated as the area difference divided by the area sum. The circle size represents the weighting factor of the individual band pair, which is mainly influenced by the vibrational band heights and the goodnesses of fit. Typically, the outer form of the pattern can be reduced to a triangle, due to symmetric duplicates.

For library search, the vibrational band area patterns of sample and reference were compared. To that end, all matching band pairs of reference and sample were extracted, and the differences of their positions and area ratios were averaged under the consideration of the weighting factors. For a high *Hit Quality Index*, the major part of all reference signals could be found within the sample pattern as well, and the differences between the area ratios were very small, which can be obtained from Fig. 9.

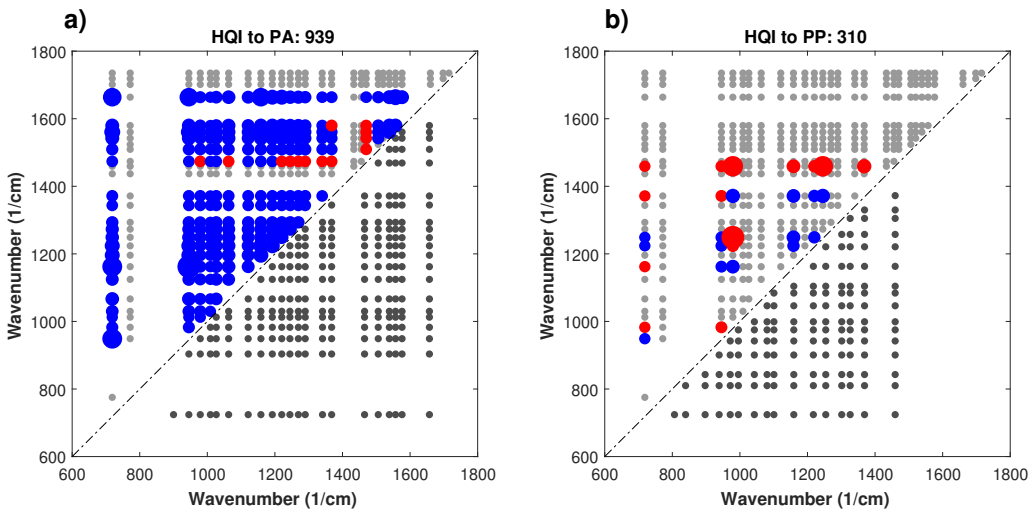


Figure S 9: Comparison of Vibrational Band Area Patterns. a) and b) show similar to Fig. 8 scatter plots of the vibrational band area patterns of a polyamide sample. However, in contrast, there are not only the triangle patterns of the sample, which can be seen above the diagonals, but there are also triangle patterns of two references below the diagonals. a) shows the comparison with a polyamide and b) with a polypropylene reference pattern. Additionally, some of the circles within the sample patterns are colored and re–sized. Individual signals that can also be found within the reference pattern and the area ratios have a similarity of at least 70 % are colored blue. If the area ratios show less similarity ($<$ 70 %) the circles are marked red. The sizes of the circles are connected to the weighting factors, in parallel to Fig. 8. Based on this data, a weighted average can be calculated that can be transformed into the HQI.

References

1. Renner, G., Schmidt, T. C. & Schram, J. A New Chemometric Approach for Automatic Identification of Microplastics from Environmental Compartments Based on FT-IR Spectroscopy. *Anal. Chem.* **89,** 12045–12053 (2017).

HQI–Microplastics Amount Relation of Case Study 2

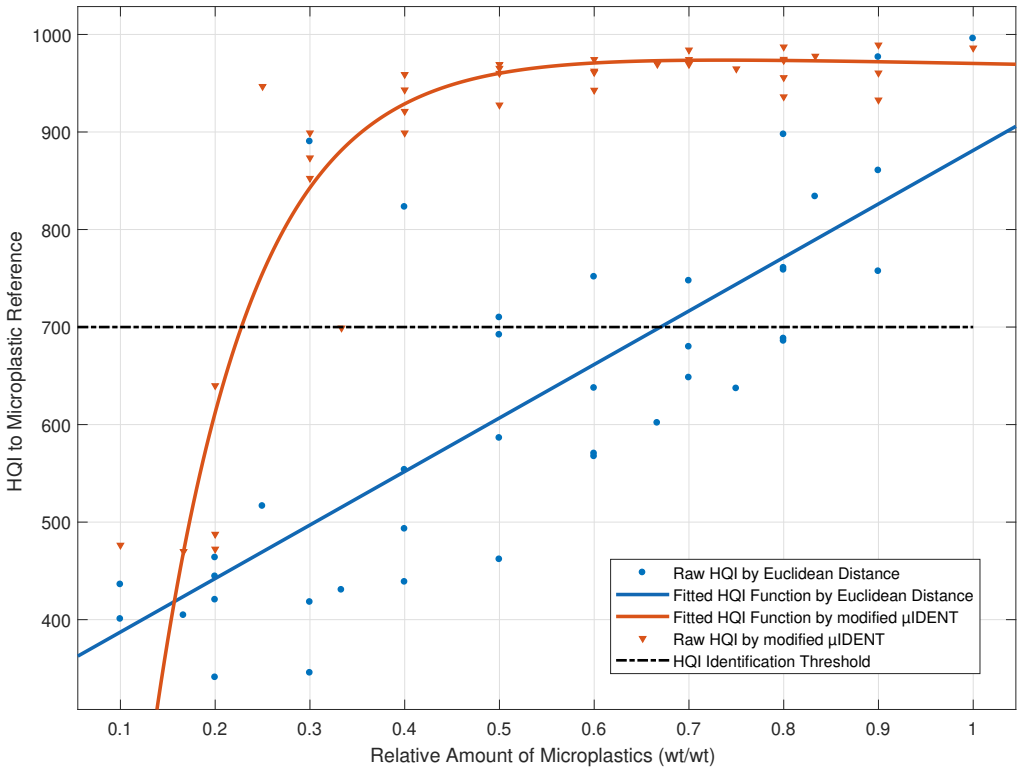


Figure S 10: Mathematical relation between the amount of microplastics and the HQI to the correct microplastic reference based on all analyzed microplastics–matrix mixtures of *Case 2*. The Euclidean Distance, as a representative of conventional library searching shows a linear relation between the two parameters, and a limit of automated identification can be observed at 66%. In contrast, the modified µIDENT approach shows a much more robust exponential relation and the limit of automated identification can be observed at 22%.

Identification Results of Case Study 3

All analyzed samples including the real samples from Case Study 3 were used to develop and validate the new microplastics identification algorithm (*modified µIDENT*). The individual identification results of Case Study 3 can be obtained from Tab. 4. They agree with other studies.[1–5]

Table S 4: Reference Identification Results of Case Study 3. All n=1117 spectra were evaluated manually by experts.

	PA	PE	PLA	PMMA	POM	PP	PS	PU	PVC
Case 3a (n = 300)	21	87	3	1	1	131	36	17	3
Case 3b (n = 817)	13	184	22	4	15	371	106	73	29

References

1. Zhu, J. *et al.* Microplastic pollution in the Maowei Sea, a typical mariculture bay of China. *Sci. Total Environ.* **658,** 62–68 (Mar. 2019).
2. Primpke, S., Lorenz, C., Rascher-Friesenhausen, R. & Gerdts, G. An automated approach for microplastics analysis using focal plane array (FPA) FTIR microscopy and image analysis. *Anal. Methods* **9,** 1499–1511 (2017).
3. Hidalgo-Ruz, V., Gutow, L., Thompson, R. C. & Thiel, M. Microplastics in the marine environment: a review of the methods used for identification and quantification. *Environ. Sci. Technol.* **46,** 3060–3075 (2012).
4. Tiwari, M., Rathod, T., Ajmal, P., Bhangare, R. & Sahu, S. Distribution and characterization of microplastics in beach sand from three different Indian coastal environments. *Mar. Pollut. Bull.* **140,** 262–273 (Mar. 2019).
5. Edo, C. *et al.* Occurrence and identification of microplastics along a beach in the Biosphere Reserve of Lanzarote. *Mar. Pollut. Bull.* **143,** 220–227 (June 2019).

Source Code of modified *µIDENT*
Create Main Window

Listing 1: python source code

```
"""
@author: Gerrit Renner
date: 2019-02-06

title: main gui
"""

# LOAD PACKAGES
# external
import tkinter
from tkinter import ttk
from PIL import Image, ImageTk
import numpy as np
import os

# internal
from mident_cfg import *
from mident_import import *
from mident_database import *
from mident_libsearch import *
from mident_mapping import *

class App():
  def __init__(self):
    self.root = tkinter.Tk()
    self.root.attributes('-fullscreen', True)
    self.root.configure(background='#333333')
    bt_close = tkinter.Button(self.root, text = 'X', command=self.quit, height=1, width=3, bg='#c54040', fg='#eeeeee', font=(None, 15, 'bold'), relief='groove')
    bt_close.pack(side='top', anchor='ne')

    # Main Frame
    self.window_width = self.root.winfo_screenwidth()
    self.window_height = self.root.winfo_screenheight()
    self.frame_main = tkinter.Frame(self.root, bg='#333333', width=self.window_width-100, height=self.window_height-50, padx=50)
    self.logo = Image.open("midentlogo.png")
    width = np.size(self.logo, axis=1)
    height = np.size(self.logo, axis=0)
    self.logo = self.logo.resize((int(width/4),int(height/4)), Image.ANTIALIAS)
    self.logo = ImageTk.PhotoImage(self.logo)
    self.logo_frame = tkinter.Label(self.root, image=self.logo,borderwidth=0).pack(side='top', anchor='n')
    self.status_text = tkinter.StringVar()
    self.status_text.set('Home')
    self.status = tkinter.Label(self.root, textvariable=self.status_text, bg='#333333', fg='#67d132', font=(None, 15, 'bold')).pack()

    # Side Menu
    self.frame_sidemenu = tkinter.Frame(self.root, width=100, height=100)
    self.frame_sidemenu.pack(side='left')
    self.frame_main.pack(side='left')
    module_array = np.array([lambda: module_database(self.frame_main, self.status_text),lambda: module_import(self.frame_main,'reference', self.status_text, self.root), lambda: module_import(self.frame_main,'sample', self.status_text, self.root), lambda: module_libsearch(self.frame_main, self.status_text, self.root), lambda: module_cfg(self.frame_main, self.status_text, self.root),lambda: module_mapping(self.frame_main, self.status_text, self.root)])
    self.create_menu(module_array)

    # define styles
    style = ttk.Style()
    style.theme_use('clam')
    style.configure('TEntry', fieldbackground='#282828', foreground='#89B594', bordercolor='#333333', lightcolor='#333333', background='#333333')
    style.configure('TLabel', background='#333333', foreground='#dddddd')
    style.configure(".", font=('Helvetica', 15), background="#333333", fieldbackground='#333333', bordercolor='#333333', lightcolor='#333333')
    style.configure("Treeview", foreground='#89B594', background='#333333', bordercolor='#333333', lightcolor='#333333')
    style.configure("Treeview.Heading", font=('Helvetica', 15, 'bold'), foreground='#FFDA8F', bordercolor='#333333', lightcolor='#333333', background='#333333', borderwidth=0)
    style.map("Treeview.Heading", background=[('focus', '#333333'),('!focus', '#333333')])
    style.configure("menu.TButton", width=80, cursor='hand2', bordercolor='#333333', lightcolor='#333333', borderwidth=0, background='#333333')
    style.map("menu.TButton",background=[('focus', '#333333'),('!focus', '#333333')])

    # check if database exists
    master_path = os.path.dirname(__file__)
    master_path = master_path[0:master_path.find("\\" + "Python" + "\\") + 8] + 'database' + '\\'
    database_master_file = master_path + 'database.csv'
    if os.path.isdir(master_path):
      if not os.path.exists(database_master_file):
        with open(database_master_file, 'w', newline='') as csvfile:
          exportdata = csv.writer(csvfile, delimiter=',')
          exportdata.writerow(['ID','TYPE','NAME','DATE','RAWDATA','RESULTS'])
    else:
      os.makedirs(master_path)
      with open(database_master_file, 'w', newline='') as csvfile:
          exportdata = csv.writer(csvfile, delimiter=',')
          exportdata.writerow(['ID','TYPE','NAME','DATE','RAWDATA','RESULTS'])

    self.root.mainloop()

  def quit(self):
    self.root.destroy()

  def create_menu(self, module):
    self.button_list = ['btmmDatabase1.png','btmmImport1.png','btmmImports1.png', 'btmmLibsearch1.png', 'btmmCfg1.png', 'btmmSmartaim1.png']
    self.button_image_array = np.array([])
    self.button_image_array2 = np.array([])
    for self.button_name in self.button_list:
      self.filename = 'icons/' + self.button_name
      self.filename2 = self.filename[:-5] + '2.png'
      self.image = Image.open(self.filename)
      self.image = self.image.resize((80, 80),Image.ANTIALIAS)
      self.image = ImageTk.PhotoImage(self.image)
      self.button_image_array = np.append(self.button_image_array, self.image)
      self.image2 = Image.open(self.filename2)
      self.image2 = self.image2.resize((80, 80),Image.ANTIALIAS)
      self.image2 = ImageTk.PhotoImage(self.image2)
      self.button_image_array2 = np.append(self.button_image_array2, self.image2)


    self.button_database = ttk.Button(self.frame_sidemenu, style='menu.TButton', image=self.button_image_array[0], command=module[0])
    self.button_database.bind("<Enter>", lambda event, h=self.button_database: h.configure(image=self.button_image_array2[0]))
    self.button_database.bind("<Leave>", lambda event, h=self.button_database: h.configure(image=self.button_image_array[0]))

    self.button_importref = ttk.Button(self.frame_sidemenu, style='menu.TButton', image=self.button_image_array[1], command=module[1])
    self.button_importref.bind("<Enter>", lambda event, h=self.button_importref: h.configure(image=self.button_image_array2[1]))
    self.button_importref.bind("<Leave>", lambda event, h=self.button_importref: h.configure(image=self.button_image_array[1]))

    self.button_importsmp = ttk.Button(self.frame_sidemenu, style='menu.TButton', image=self.button_image_array[2], command=module[2])
    self.button_importsmp.bind("<Enter>", lambda event, h=self.button_importsmp: h.configure(image=self.button_image_array2[2]))
    self.button_importsmp.bind("<Leave>", lambda event, h=self.button_importsmp: h.configure(image=self.button_image_array[2]))

    self.button_libsearch = ttk.Button(self.frame_sidemenu, style='menu.TButton', image=self.button_image_array[3], command=module[3])
    self.button_libsearch.bind("<Enter>", lambda event, h=self.button_libsearch: h.configure(image=self.button_image_array2[3]))
    self.button_libsearch.bind("<Leave>", lambda event, h=self.button_libsearch: h.configure(image=self.button_image_array[3]))

    self.button_cfg = ttk.Button(self.frame_sidemenu, style='menu.TButton', image=self.button_image_array[4], command=module[4])
    self.button_cfg.bind("<Enter>", lambda event, h=self.button_cfg: h.configure(image=self.button_image_array2[4]))
    self.button_cfg.bind("<Leave>", lambda event, h=self.button_cfg: h.configure(image=self.button_image_array[4]))

    self.button_mapping = ttk.Button(self.frame_sidemenu, style='menu.TButton', image=self.button_image_array[5], command=module[5])
    self.button_mapping.bind("<Enter>", lambda event, h=self.button_mapping: h.configure(image=self.button_image_array2[5]))
    self.button_mapping.bind("<Leave>", lambda event, h=self.button_mapping: h.configure(image=self.button_image_array[5]))

    self.button_importref.pack()
    self.button_importsmp.pack()
    self.button_database.pack()
    self.button_libsearch.pack()
    self.button_cfg.pack()
    self.button_mapping.pack()

root = App()
```

Source Code of modified *µIDENT*
Import Module

Listing 2: python source code

```
"""
@author: Gerrit Renner
date: 2019-02-06

title: import module
"""

# LOAD PACKAGES
# external
from tkinter.filedialog import askopenfilenames

# internal
from mident_job import *

class module_import:
  def __init__(self, master, type_, status, main):
    for widget in master.winfo_children():
      widget.destroy()
    fname = askopenfilenames(filetypes=(("Shimadzu AIM 9000", "*.apit"),
                        ("Shimadzu AIM 9000", "*.amap"),
                        ("CSV files", "*.csv"),
                        ("TXT files", "*.txt")))
    if fname:
      module_job(master, fname, type_, status, main)
```

Source Code of modified *µIDENT*
Job Module

Listing 3: python source code

```
"""
@author: Gerrit Renner
date: 2019-02-06

title: job module
"""

# LOAD PACKAGES
# external
from dateutil.parser import parse
from tkinter import ttk
import tkinter

#internal
from mident_fit import *

class module_job:
  def __init__(self, master, fnames, type_, status, main):
    for widget in master.winfo_children():
      widget.destroy()
    # get entries
    items = []
    for item in fnames:
      tmp_type = type_
      tmp_name, tmp_date = self.get_filename(item)
      tmp_item = (tmp_type, tmp_name, tmp_date)
      items.append(tmp_item)
    # create table
    table_header = ['TYPE', 'FILE NAME', 'DATE']
    column_widths = (150, 300, 200)
    self.tree = ttk.Treeview(master = master, columns=table_header, show="headings", height
    =20)
    self.tree.grid(column=0, row=1, sticky='nsew', in_=master)
    master.grid_columnconfigure(0, weight=1)
    master.grid_rowconfigure(0, weight=1)
    # set headings
    for ix in enumerate(table_header):
      self.tree.heading(table_header[ix[0]], text=table_header[ix[0]], anchor='w')
      self.tree.column(table_header[ix[0]], width=column_widths[ix[0]])
    # set items
    for item in items:
      self.tree.insert('', 'end', values=item)

    self.tree.bind("<Double-1>", self.OnDoubleClick)
    # create edit frame
    self.edit_frame = tkinter.Frame(master, width=670, height=200, bg='#333333')
    self.edit_frame.grid(row=0, column=0)
    # create run button
    self.run_button = tkinter.Button(master, text='Start Curve Fitting',
                    font=(None, 15, 'bold'), width=55,
                    relief='groove', bg='#317256',
                    fg='#dddddd', command= lambda: module_fit(master, fnames,items, status,
    main))
    self.run_button.grid(row=2, column=0, sticky='n')

  def get_filename(self, filename):
    # delete path
    flag = 0
    while flag == 0:
      ix = filename.find('/')
      if ix != -1:
        filename = filename[ix+1:]
      else:
        flag = 1
    # delete ending
    ix = filename.find('.')
    filename = filename[0:ix]
    # extract date
    try:
      ix = filename.find('_')
      date = filename[0:ix]
      try:
        tmp = int(date[0])
        month = ''
        for i in range(len(date)):
          try:
            tmp = int(date[i])
          except:
            month = month + date[i]
        ix2 = date.find(month)
        _len = len(month)
        if month == 'i' or month == 'I':
          month = '01'
        elif month == 'ii' or month == 'II':
          month = '02'
        elif month == 'iii' or month == 'III':
          month = '03'
        elif month == 'iv' or month == 'IV':
          month = '04'
        elif month == 'v' or month == 'V':
          month = '05'
        elif month == 'vi' or month == 'VI':
          month = '06'
        elif month == 'vii' or month == 'VII':
          month = '07'
        elif month == 'viii' or month == 'VIII':
          month = '08'
        elif month == 'ix' or month == 'IX':
          month = '09'
        elif month == 'x' or month == 'X':
          month = '10'
        elif month == 'xi' or month == 'XI':
          month = '11'
        elif month == 'xii' or month == 'XII':
          month = '12'
        if ix2 > 0:
          year = date[0:ix2]
          day = date[ix2+_len:]
        else:
          if len(date) == 6:
            year = '20' + date[0:2]
            month = date[2:4]
            day = date[4:6]
          else:
            year = '2015'
            month = '08'
            day = '01'

        date = str(year) + '-' + month + '-' + str(day)
        name = filename[ix+1:]
        flag = False
      except:
        date = '2015-8-1'
        name = filename
        flag = True
    except:
      date = '2015-8-1'
      name = filename
      flag = True
    if flag:
      try:
        date = parse(filename[0:8], yearfirst=True)
        date = str(date.year) + '-' + str(date.month) + '-' + str(date.day)
        name = filename[9:]
      except ValueError as err:
        date = '2015-8-1'
        name = filename
    return name, date


  def OnDoubleClick(self, event):
    item = self.tree.selection()[0]
    self.Edit(item)

  def Edit(self, item):
    for widget in self.edit_frame.winfo_children():
      widget.destroy()
    # read item
    item_dict = self.tree.item(item)
    item_ID = item
    item = item_dict.get('values')
    # set labels
    self.type_lbl = ttk.Label(self.edit_frame,text='type: ',
                    font=(None, 15, 'bold')).grid(row=1, sticky='e')
    self.name_lbl = ttk.Label(self.edit_frame,text='file name: ',
                    font=(None, 15, 'bold')).grid(row=2, sticky='e')
    self.date_lbl = ttk.Label(self.edit_frame,text='date: ',
                    font=(None, 15, 'bold')).grid(row=3, sticky='e')
    # set entries
    self.entry_type = ttk.Entry(self.edit_frame, font=(None, 15, 'bold'))
    self.entry_name = ttk.Entry(self.edit_frame, font=(None, 15, 'bold'))
    self.entry_date = ttk.Entry(self.edit_frame, font=(None, 15, 'bold'))
    # set buttons
    self.apply = tkinter.Button(self.edit_frame, text='Apply', font=(None, 15, 'bold'),
                  command= lambda: self.Update_Job(item_ID),
                  relief='groove', bg='#317256', fg='#dddddd')
    self.delete = tkinter.Button(self.edit_frame, text='Delete Sample', font=(None, 15, 'bold'
    ),
                  command= lambda: self.Update_Job(item_ID),
                  relief='groove', bg='#c54040', fg='#dddddd')
    # preset entry values
    self.entry_type.insert(0,item[0])
    self.entry_name.insert(0,item[1])
    self.entry_date.insert(0,item[2])
    # place entries
    self.entry_type.grid(row=1, column=1)
    self.entry_name.grid(row=2, column=1)
    self.entry_date.grid(row=3, column=1)
    # place buttons
    self.apply.grid(row=1, column=2, rowspan=3)
    self.delete.grid(row=1, column=3, rowspan=3)
    # set placeholder for minimum frame size
    self.placeholder_width = tkinter.Frame(self.edit_frame, bg='#333333', width=670, height
    =55)
    self.placeholder_width.grid(row=0, column=0, columnspan=4)
    self.placeholder_width = tkinter.Frame(self.edit_frame, bg='#333333', width=670, height
    =55)
    self.placeholder_width.grid(row=4, column=0, columnspan=4)

  def Update_Job(self, item):
    new_value = (self.entry_type.get(),
          self.entry_name.get(),
          self.entry_date.get())
    for widget in self.edit_frame.winfo_children():
      widget.destroy()
    self.tree.item(item, values=new_value)
```

Source Code of modified *μIDENT*
Curve Fit Module

Listing 4: python source code

```python
"""
@author: Gerrit Renner
date: 2019-02-06

title: fit module
"""

# LOAD PACKAGES
# external
import time
from scipy.optimize import curve_fit, minimize
import numpy as np
import scipy.integrate as integrate
import csv
import os
import pandas as pd
from matplotlib.figure import Figure
import matplotlib
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2TkAgg

# internal
import mident
from mident_libsearch import *
import importApit

class module_fit:
  def __init__(self, master, fnames, items, status, main):
    for widget in master.winfo_children():
      widget.destroy()
    self.fit_que(fnames,master, items, status, main)

  def fit_que(self, fnames,master, items, status, main):
    start_time = time.time()
    for i in range(len(fnames)):
      for widget in master.winfo_children():
        widget.destroy()
      if i == len(fnames) - 1:
        libsearch = False
      else:
        libsearch = False
      self.fit_spectrum(fnames[i],master,items[i],libsearch)
      av_time_fit = (time.time() - start_time) / (i+1)
      time_to_go = (len(fnames) - i - 1) * av_time_fit

      print('Average time per fit: ' + str(round(av_time_fit)) + ' seconds')
      print('Time till Job is done: ' + str(round(time_to_go)) + ' seconds')
      print('_________ ' + str(i + 1) + ' of ' + str(len(fnames)) + ' _____')
      status.set('(' + str(int((i+1)/len(fnames)*100)) + ' %), Spectrum ' + str(i + 1) + ' of ' + str(len(fnames)) + ', Time till Job is done: ' + str(round(time_to_go)) + ' seconds')
      main.update()
    status.set('(' + str(int((i+1)/len(fnames)*100)) + ' %), Spectrum ' + str(i + 1) + ' of ' + str(len(fnames)) + ', Time: ' + str(round((time.time() - start_time))) + ' seconds')
    main.update()


  def fit_spectrum(self,spectrum,master,items,libsearch=False):
    # load cfg file
    '''___Master Path__________________________________________________'''
    master_path = os.path.dirname(__file__)
    master_path = master_path[0:master_path.find("\\" + "Python" + "\\") + 8] #+ 'database' + '\\'
    cfg_master_file = master_path + 'mident_config.nfo'
    with open(cfg_master_file) as csvfile:
      cfg_master_csv = csv.reader(csvfile, delimiter='~')
      cfg = []
      for row in cfg_master_csv:
        cfg.append(row)
    cfg = cfg[0]
    cfg_dict = {}
    for i in range(len(cfg)):
      cfg_dict[cfg[i][1:cfg[i].find(' = ')]] = cfg[i][cfg[i].find(' = ')+3:-1]
    '''___DEBUG MODE___________________________________________________'''
    debug_mode = False
    '''___IMPORT MODE__________________________________________________'''
    import_mode = False
    '''___WAVELET DENOISE______________________________________________'''
    wavelet_denoise = False
    '''___Master Path__________________________________________________'''
    master_path = os.path.dirname(__file__)
    master_path = master_path[0:master_path.find("\\" + "Python" + "\\") + 8] + 'database' + '\\'
    '''___Import Data__________________________________________________'''
    for i in range(0,len(spectrum)):
      if spectrum[i] == '.':
        flag = i
    data_type = spectrum[flag+1:]
    if data_type == 'amap':
      data_type = 'apit'
    if data_type == 'apit':
      if import_mode:
        try:
          x,y,table_x,table_y = importApit.import_apit(spectrum, calc_mean=False, KK_transform=False, cut=True, extend=True, mapping=False)
        except:
          x,y = importApit.import_apit(spectrum, calc_mean=False, KK_transform=False, cut=True, extend=True, mapping=False)
      else:
        try:
          x,y,table_x,table_y = importApit.import_apit(spectrum, calc_mean=False, KK_transform=False, cut=True, extend=True, mapping=True)
        except:
          x,y = importApit.import_apit(spectrum, calc_mean=False, KK_transform=False, cut=True, extend=True, mapping=True)
      flag = 1

    if data_type == 'txt':
      print('load txt file')
      x = []
      y = []
      with open(spectrum, newline='') as csvfile:
        data = csv.reader(csvfile, delimiter='\t')
        for row in data:
          if len(row) > 1:
            row[0] = row[0].replace(',','.')
            row[1] = row[1].replace(',','.')
            x = np.append(x, float(row[0]))
            y = np.append(y, float(row[1]))
      y = y[(x>700) == (x<1850)]
      x = x[(x>700) == (x<1850)]
      flag = 1

    if data_type == 'csv':
      print('load csv file')
      with open(spectrum, newline='') as csvfile:
        data = csv.reader(csvfile,delimiter=',')
        i = 0
        for row in data:
          i = i+1
          if i == 1:
            x = [float(i) for i in row]
            x = np.array(x)
          if i == 2:
            y = [float(i) for i in row]
            y = np.array(y)
      y = y[(x>700) == (x<2200)]
      x = x[(x>700) == (x<2200)]
      flag = 1
    print('loading complete')

    if import_mode:
      print('only import')
      '''___Baseline Correction__________________________________________'''
      number_of_measurements = np.shape(y)
      try:
        number_of_measurements = number_of_measurements[1]
      except:
        number_of_measurements = 1
      print('baseline correction complete')
      '''___Export_______________________________________________________'''
      for i in range(number_of_measurements):
        filename_raw = master_path + items[1] + '_' + str(i+1) + '_raw.csv'
        with open(filename_raw, 'w', newline='') as csvfile:
          exportdata = csv.writer(csvfile, delimiter=',')
          exportdata.writerow(x)
          exportdata.writerow(y[:,i])
      try:
        filename_raw = master_path + items[1] + '_' + str(i+1) + '_pos.csv'
        with open(filename_raw, 'w', newline='') as csvfile:
          exportdata = csv.writer(csvfile, delimiter=',')
          exportdata.writerow(np.array([table_x,table_y]))
      except:
        pass
    else:
      '''___Check if Spectrum is relevant________________________________'''
      df = pd.DataFrame(y)
      y_smooth = df.rolling(25,center=True,min_periods=1).mean()
      y_movingAV = df.rolling(500,center=True,min_periods=1).mean()
      criterion = np.mean(((y_smooth - y_movingAV)**2)**.5)
      criterion[0] = 1
      if criterion[0] < .06:
        print('no relevant data loaded')
      else:
        y = y[x < 2200]
        x = x[x < 2200]
        '''___Smoothing ___________________________________________________'''
        y_smooth = mident.ir_smooth(x,y)
        yOLD = y
        y = y_smooth
        '''___Baseline Correction__________________________________________'''
        baseline = mident.airPLS(y,lambda_=80,porder=2)
        df = pd.DataFrame(baseline)
        baseline_smooth = df.rolling(100,center=True,min_periods=1).mean()
        baseline = np.min([baseline_smooth.values[:,0],baseline],axis=0)
        y_baseline = y - baseline
        print('baseline correction complete')
        '''___Correct Spikes_______________________________________________'''
        if wavelet_denoise:
          # Wavelet Denoise
          y_den_wavelet = mident.waveletSmooth(y_baseline)
          # Water Denoise
          baseline3 = mident.waveletSmooth(y_den_wavelet,level=3)
          _weighting_vector = (erf((x-1400)*.3)+1)/2
          try:
            y_den_wavelet = y_den_wavelet[:-1] * (1-_weighting_vector) + baseline3[:-1] * _weighting_vector
          except:
            y_den_wavelet = y_den_wavelet[:-1] * (1-_weighting_vector[:-1]) + baseline3[:-1] * _weighting_vector[:-1]
          y_den_wavelet = np.append(y_den_wavelet,0)
        else:
          y_den_wavelet = np.append(y_baseline,0)
        '''___Initialise Peaks_____________________________________________'''
        parameter = mident.peak_init(x, y_den_wavelet[:-1],y_baseline)
        #exit()
        print('peak initialize complete')
        '''___Curve Fit____________________________________________________'''
        # cut spectrum
        xold = x
        yold = y
        y = y_baseline
        parameter_broad = parameter.copy()
        parameter_broad[2::5] = parameter_broad[2::5]*1.5
        y_init = mident.apv_func(xold,parameter_broad)
        y_init2 = y_init
        # min number of points that are higher than threshold
        min_number = 10#5
        threshold = .002
        # binary filter
        binary_filter = np.min([y, y_init],axis=0) > threshold
        flag_start = 0
        cut_start = []
        for i in range(0,len(y) - min_number):
          if flag_start == 0:
            if sum(binary_filter[i:i + min_number]) == min_number:
              flag_start = 1
              cut_start = np.append(cut_start, xold[i])

          if flag_start == 1:
            if binary_filter[i] == False:
              flag_start = 0
              cut_start = np.append(cut_start, xold[i])
        if len(cut_start) % 2 != 0:
          cut_start = np.append(cut_start,max(xold))
        x_cut = np.reshape(cut_start,[int(len(cut_start)/2),2])
        n_cut = np.size(x_cut,0)
        print('Cut complete')
        # parameter
        p_position = parameter[0::5]
        p_height = parameter[1::5]
        p_width = parameter[2::5]
        p_shape = parameter[3::5]
        p_asym = parameter[4::5]
        parameter_fitted = []
        parameter_err = []
        # reduce data to significant datapoints
        x_relevant = np.array([])
        y_relevant = np.array([])
        for i in range(len(p_position)):
          # first peak
          if i == 0:
            tmp_start = xold[0]
            tmp_end = p_position[i+1]
          # last peak
          if i == len(p_position)-1:
            tmp_start = p_position[i-1]
            tmp_end = xold[-1]
          # other peaks
          if (i>0)*(i<len(p_position)-1):
            tmp_start = p_position[i-1]
            tmp_end = p_position[i+1]
          x_tmp = xold[(xold>=tmp_start)*(xold<=tmp_end)]
          y_tmp = y[(xold>=tmp_start)*(xold<=tmp_end)]
          x_center = x_tmp[np.argmin(np.abs(x_tmp-p_position[i]))]
          y_center = y_tmp[np.argmin(np.abs(x_tmp-p_position[i]))]
          x_min_left = x_tmp[np.argmin(y_tmp[x_tmp<=x_center])]
          x_min_right = x_tmp[np.argmin(y_tmp[x_tmp>=x_center])+len(y_tmp[x_tmp<x_center])]

          x_left = x_tmp[(x_tmp>=x_min_left)*(x_tmp<=x_center)]
          y_left = y_tmp[(x_tmp>=x_min_left)*(x_tmp<=x_center)]
          y_left_norm = (y_left - min(y_left))/(max(y_left - min(y_left)))
          x_left50 = x_left[np.argmin(abs(y_left_norm - .5))]
          y_left50 = y_left[np.argmin(abs(y_left_norm - .5))]
          x_left25 = x_left[np.argmin(abs(y_left_norm - .25))]
          y_left25 = y_left[np.argmin(abs(y_left_norm - .25))]

          x_right = x_tmp[(x_tmp>=x_center)*(x_tmp<=x_min_right)]
          y_right = y_tmp[(x_tmp>=x_center)*(x_tmp<=x_min_right)]
          y_right_norm = (y_right - min(y_right))/(max(y_right - min(y_right)))
          x_right50 = x_right[np.argmin(abs(y_right_norm - .5))]
          y_right50 = y_right[np.argmin(abs(y_right_norm - .5))]
          x_right25 = x_right[np.argmin(abs(y_right_norm - .25))]
          y_right25 = y_right[np.argmin(abs(y_right_norm - .25))]

          x_tmp,_index = np.unique(np.array([x_left[0], x_left25, x_left50, x_center, x_right50, x_right25, x_right[-1]]),return_index=True)
          y_tmp = np.array([y_left[0], y_left25, y_left50, y_center, y_right50, y_right25, y_right[-1]])
          y_tmp = y_tmp[_index]
          x_relevant = np.append(x_relevant, x_tmp)
          y_relevant = np.append(y_relevant, y_tmp)

        x_relevant,_index = np.unique(x_relevant,return_index=True)
        y_relevant = y_relevant[_index]

        for i in range(n_cut):
          x_tmp = xold[(xold>=x_cut[i,0])==(xold<=x_cut[i,1])] #
          y_tmp = y[(xold>=x_cut[i,0])==(xold<=x_cut[i,1])] #
          tmp_p_position = p_position[(p_position>=x_cut[i,0])==(p_position<=x_cut[i,1])]
          tmp_p_height = p_height[(p_position>=x_cut[i,0])==(p_position<=x_cut[i,1])]
          tmp_p_width = p_width[(p_position>=x_cut[i,0])==(p_position<=x_cut[i,1])]
          tmp_p_shape = p_shape[(p_position>=x_cut[i,0])==(p_position<=x_cut[i,1])]
          tmp_p_asym = p_asym[(p_position>=x_cut[i,0])==(p_position<=x_cut[i,1])]
          tmp_parameter = np.append(tmp_p_position, tmp_p_height)
          tmp_parameter = np.append(tmp_parameter, tmp_p_width)
          tmp_parameter = np.append(tmp_parameter, tmp_p_shape)
          tmp_parameter = np.append(tmp_parameter, tmp_p_asym)
          tmp_parameter = np.reshape(tmp_parameter,[5,int(len(tmp_parameter)/5)])
          tmp_parameter = np.reshape(np.transpose(tmp_parameter),[np.size(tmp_parameter),1])
          # limits
          lo = np.zeros([len(tmp_parameter),1])
          lo[0::5] = tmp_parameter[0::5] - 10
          lo[1::5] = tmp_parameter[1::5] * .5
          lo[2::5] = tmp_parameter[2::5] * .2
          lo[3::5] = 0
          lo[4::5] = -.9 / tmp_parameter[2::5]

          up = np.zeros([len(tmp_parameter),1])
          up[0::5] = tmp_parameter[0::5] + 10
          up[1::5] = tmp_parameter[1::5] * 10 + 1
          up[2::5] = tmp_parameter[2::5] * 2 + 1
          up[3::5] = 1
          up[4::5] = .9 / tmp_parameter[2::5]
          if len(tmp_parameter[:,0]) != 0:
            try:
              yfit = curve_fit(mident.apv_fit,x_tmp,y_tmp,p0=tmp_parameter[:,0],bounds=[lo[:,0],up[:,0]],method='trf',maxfev=1000)
              # parameter errors
              err_ = []
              for i in range(len(yfit[0])):
                try:
                  err_ = np.append(err_, np.absolute(yfit[1][i,i])**0.5)
                except:
                  err_ = np.append(err_,0)
              # check for negative peaks
              tmp_heights = yfit[0]
              tmp_heights = tmp_heights[1::5]
              while sum(tmp_heights < 0) > 0:
                _filter = np.repeat(tmp_heights > 0,5)
                tmp_parameter = tmp_parameter[_filter]
                lo = lo[_filter,0]
                up = up[_filter,0]
                yfit = curve_fit(mident.apv_fit,x_tmp,y_tmp,p0=tmp_parameter[:,0],bounds=[lo[:,0],up[:,0]],method='trf',maxfev=1000)
                tmp_heights = yfit[0]
                tmp_heights = tmp_heights[1::5]
              parameter_fitted = np.append(parameter_fitted, yfit[0])
              parameter_err = np.append(parameter_err, err_)
            except:
              try:
                tmp_parameter2 = np.append(tmp_parameter[:,0],tmp_parameter[:,0])
                lo = np.append(lo[:,0],lo[:,0])
                up = np.append(up[:,0],up[:,0])
                yfit = curve_fit(mident.apv_fit,x_tmp,y_tmp,p0=tmp_parameter2,bounds=[lo,up],method='trf',maxfev=1000)

                # parameter errors
                err_ = []
                for i in range(len(yfit[0])):
                  try:
                    err_ = np.append(err_, np.absolute(yfit[1][i,i])**0.5)
                  except:
                    err_ = np.append(err_,0)
                parameter_fitted = np.append(parameter_fitted, yfit[0])
                paramter_err = np.append(parameter_err, err_)
                print('re-fitted')
              except:
                print('sorry no fit')
                parameter_fitted = np.append(parameter_fitted, tmp_parameter[:,0])
                paramter_err = np.append(parameter_err, np.ones(len(tmp_parameter[:,0]))*np.Inf)
        yfit = mident.apv_func(xold,parameter_fitted)
        print('Fit complete')
        # sort parameter
        _tmp_position = parameter_fitted[0::5]
        _tmp_height = parameter_fitted[1::5]
        _tmp_width = parameter_fitted[2::5]
        _tmp_shape = parameter_fitted[3::5]
        _tmp_asym = parameter_fitted[4::5]
        _index = sorted(range(len(_tmp_position)),key=lambda x:_tmp_position[x])
        _tmp_position = _tmp_position[_index]
        _tmp_height = _tmp_height[_index]
        _tmp_width = _tmp_width[_index]
        _tmp_shape = _tmp_shape[_index]
        _tmp_asym = _tmp_asym[_index]
        # check for out-of-range peaks
        _oor_filter = (_tmp_position < min(xold)) + (_tmp_position > max(xold))

        _parameter = np.zeros(sum(~_oor_filter)*5)#parameter_fitted * 0
        _parameter[0::5] = _tmp_position[~_oor_filter]
        _parameter[1::5] = _tmp_height[~_oor_filter]
        _parameter[2::5] = _tmp_width[~_oor_filter]
        _parameter[3::5] = _tmp_shape[~_oor_filter]
        _parameter[4::5] = _tmp_asym[~_oor_filter]
        parameter_fitted = _parameter

        # check for spikes
        _spike_filter = parameter_fitted[2::5] > 2
        parameter_fitted = parameter_fitted[np.repeat(_spike_filter,5)]

        peak_blacklist = [1]
        flag = 0
        # Delete Bad Peaks
        TPOS = parameter_fitted[0::5]
        err = np.zeros(int(len(parameter_fitted)/5))
        for i in range(int(len(parameter_fitted)/5)):
          tmp_param = parameter_fitted[i*5:(i+1)*5]
          x_tmp = xold[(xold > tmp_param[0]-tmp_param[2])*(xold < tmp_param[0] + tmp_param[2])]
          y_tmp_fit = yfit[(xold > tmp_param[0]-tmp_param[2])*(xold < tmp_param[0] + tmp_param[2])]
          y_tmp_raw = y[(xold > tmp_param[0]-tmp_param[2])*(xold < tmp_param[0] + tmp_param[2])]
          err[i] = np.mean(abs(y_tmp_fit - y_tmp_raw)) / tmp_param[1]
        filter_bad = err > 1
        filter_bad = np.repeat(filter_bad,5)
        baseline_new = mident.apv_func(xold,parameter_fitted[filter_bad])
        baseline = baseline + baseline_new
        parameter_fitted = parameter_fitted[~filter_bad]

        # check for out-of-range peaks
        while (len(peak_blacklist) > 0) + (flag == 2):
          _peak_index = mident.peak_combination(xold,parameter_fitted,baseline)
          print('Peak Overlay complete')

          '''___Results______________________________________________________'''
          i = 0
          result_position = []
          result_area = []
          result_significance = []
          result_height = []
          result_width = []
          _index_lin = np.linspace(0,len(_peak_index)-1,len(_peak_index),dtype=int)
          peak_blacklist = []

          while i < len(_peak_index):
            actual_peak_index = _peak_index[i]

            # check for grouping peaks
            actual_peaks = _index_lin[_peak_index == actual_peak_index]

            tmp_parameter = parameter_fitted[i*5:actual_peaks[-1]*5+5]
            y_tmp = mident.apv_func(xold,tmp_parameter)
            y_tmp2 = mident.apv_func(xold,parameter_fitted)

            tmp_position = tmp_parameter[0::5]
            tmp_heights = tmp_parameter[1::5]
            tmp_position = tmp_position[np.argmax(tmp_heights)]
            tmp_position = xold[np.argmax(y_tmp)]
            tmp_width = tmp_parameter[2::5]
            tmp_area = np.trapz(y_tmp,xold,dx=0.1)
            tmp_filter = y_tmp / max(y_tmp) > .1
            # filter right side
            right_peak = max(_index_lin[_peak_index == actual_peak_index]) + 1
            if right_peak <= max(_index_lin):
              right_position = parameter_fitted[right_peak*5]
              right_tmp_y = y_tmp2[(xold < right_position) * (xold >= tmp_position)]
              right_tmp_x = xold[(xold < right_position) * (xold >= tmp_position)]
              right_tmp_x = right_tmp_x[np.argmin(right_tmp_y)]
              right_filter = xold <= right_tmp_x
            else:
              right_filter = xold <= max(xold)
            # filter left side
            left_peak = min(_index_lin[_peak_index == actual_peak_index]) - 1
            if left_peak >= 0:
              left_position = parameter_fitted[left_peak*5]
              left_tmp_y = y_tmp2[(xold > left_position) * (xold <= tmp_position)]
              left_tmp_x = xold[(xold > left_position) * (xold <= tmp_position)]
              left_tmp_x = left_tmp_x[np.argmin(left_tmp_y)]
              left_filter = xold >= left_tmp_x
            else:
              left_filter = xold <= max(xold)
            tmp_filter2 = left_filter * right_filter * tmp_filter
            if len(y[tmp_filter2]) > 3:
              tmp_significance = max(y[tmp_filter2]) / (sum((y[tmp_filter2] - y_tmp2[tmp_filter2])**2) / sum((y[tmp_filter2] - np.mean(y[tmp_filter2]))**2))**.5
              result_position = np.append(result_position,tmp_position)
              result_area = np.append(result_area,tmp_area)
              result_significance = np.append(result_significance,tmp_significance)
              result_height = np.append(result_height,max(tmp_heights))
              result_width = np.append(result_width, sum(tmp_width))
            else:
              tmp_significance = 0
            i = i + sum(_peak_index == actual_peak_index)
            if tmp_significance < 1e-4 or tmp_significance/tmp_area < 0.01:
              pass

          # Delete black listed peaks
          for i in peak_blacklist:
            parameter_fitted[int(i)*5:int(i)*5+5] = np.nan

          parameter_fitted = parameter_fitted[~np.isnan(parameter_fitted)]
          flag = flag + 1

        yfit = mident.apv_func(xold,parameter_fitted)
        print('Results complete')
        '''___Export_______________________________________________________'''
        if items[0] == 'reference':
          filetype = '_REF'
        else:
          filetype = '_SMP'
        filename = master_path + items[1] + filetype + '_results.csv'
        filename_raw = master_path + items[1] + filetype + '_raw.csv'
        filename_fit = master_path + items[1] + filetype + '_fit.csv'
        database_master_file = master_path + 'database.csv'
        with open(database_master_file) as csvfile:
          database_master_csv = csv.reader(csvfile, delimiter=',')
          database_ID = []
          database_TYPE = []
          database_NAME = []
          database_DATE = []
          database_RAWDATA = []
          database_RESULTS = []
          for row in database_master_csv:
            if len(row) > 0:
              database_ID.append(row[0])
              database_TYPE.append(row[1])
              database_NAME.append(row[2])
              database_DATE.append(row[3])
              database_RAWDATA.append(row[4])
              database_RESULTS.append(row[5])
        # check if entry already exists
        flag = 0
        for i in range(0,len(database_NAME)):
          if database_NAME[i] == items[1] and database_DATE[i] == items[2]:
            flag = i

        if flag > 0:
          print('Entry exists')
        else:
          print('Entry is new')
          if len(database_ID) > 1:
            database_ID.append(int(database_ID[-1]) + 1)
          else:
            database_ID.append(10000+len(database_ID))
          database_TYPE.append(items[0])
          database_NAME.append(items[1])
          database_DATE.append(items[2])
          database_RAWDATA.append(filename_raw)
          database_RESULTS.append(filename)
          # export RAW File
          with open(filename_raw, 'w', newline='') as csvfile:
            exportdata = csv.writer(csvfile, delimiter=',')
            exportdata.writerow(xold)
            exportdata.writerow(yOLD)
          with open(database_master_file, 'w', newline='') as csvfile:
            exportdata = csv.writer(csvfile, delimiter=',')
            for row in range(0,len(database_NAME)):
              exportdata.writerow([database_ID[row], database_TYPE[row], database_NAME[row], database_DATE[row], database_RAWDATA[row], database_RESULTS[row]])

        with open(filename, 'w', newline='') as csvfile:
          exportdata = csv.writer(csvfile, delimiter=',')
          exportdata.writerow(np.transpose(result_position))
          exportdata.writerow(np.transpose(result_area))
          exportdata.writerow(np.transpose(result_significance))
          exportdata.writerow(np.transpose(parameter_fitted))
          exportdata.writerow(np.transpose(result_width))
          try:
            exportdata.writerow(np.array([table_x, table_y]))
          except:
            pass

        with open(filename_fit, 'w', newline='') as csvfile:
            exportdata = csv.writer(csvfile, delimiter=',')
            exportdata.writerow(xold)
            exportdata.writerow(yfit)
            exportdata.writerow(yOLD)
            exportdata.writerow(baseline)

        print('data export complete')
        '''___Plot Data____________________________________________________'''
        # get frame size
        parentname = master.winfo_parent()
        parent = master._nametowidget(parentname)
        width = parent.winfo_width()-100
        height = parent.winfo_height()-50

        # get peak heights
        peak_heights = []
        for pos in result_position:
          tmp_difference = abs(pos - xold)
          tmp_index = np.argmin(tmp_difference)
          peak_heights = np.append(peak_heights,(y[tmp_index]+baseline[tmp_index])*1.07)

        # get relative peak significances
        rel_peak_significance = result_significance/np.sum(result_significance)*100


        # create figure
        f = Figure(figsize=(width/100,height/100),
              dpi=100, facecolor='#333333')
        a = f.add_subplot(111,facecolor='#333333')
        for child in a.get_children():
          if isinstance(child, matplotlib.spines.Spine):
            child.set_color('#bbbbbb')
        a.set_xlabel(cfg_dict.get('xunit'),fontweight='bold', fontsize=16)
        a.set_ylabel(cfg_dict.get('yunit'),fontweight='bold', fontsize=16)
        a.xaxis.label.set_color('#bbbbbb')
        a.yaxis.label.set_color('#bbbbbb')
        a.tick_params(axis='x', colors='#bbbbbb')
        a.tick_params(axis='y', colors='#bbbbbb')
        a.invert_xaxis()

        a.plot([xold[0],xold[0]],[min(yfit+baseline),max(yfit+baseline)*1.3],lw=0)
        a.plot(xold, yfit+baseline,lw=8,color='#EE3135')
        a.scatter(xold,yOLD, marker='+', color='#999999')
        a.scatter(result_position,peak_heights,marker="v",color='#EE3135')
        # plot reference
        try:
          a.plot(x_ref, y_ref, lw=1, color='#FFAA00',linestyle='--')
          res_str = "assigned as: " + reference_names[results] + ', HQI = ' + str(hqi)
          a.set_title(res_str , position = (0.5,0.9), fontweight='bold', fontsize=16, color='#bbbbbb')
        except:
          pass
        # create annotations
        for i in range(0,len(result_position)):
          a.text(result_position[i], peak_heights[i]*1.01, str(i+1), fontsize=15,color='white', rotation=45, horizontalalignment='left', verticalalignment='bottom')
        # calculate normalized root mean squared error
        rmse = (np.mean((yfit+baseline - yOLD)**2))**.5 / (np.max(yOLD) - np.min(yOLD))
        print(rmse)
        f.set_tight_layout(True)
        canvas = FigureCanvasTkAgg(f, master)
        canvas.draw()
        canvas.get_tk_widget().pack(side=tkinter.BOTTOM, fill=tkinter.BOTH, expand=True)
        canvas._tkcanvas.pack(side=tkinter.TOP, fill=tkinter.BOTH, expand=True)
        print('plot complete')
```

Source Code of modified *µIDENT*
Database Module

Listing 5: python source code

```
"""
@author: Gerrit Renner
date: 2019-02-06

title: database module
"""

# LOAD PACKAGES
# external
import os
import csv
from tkinter import ttk
import tkinter
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib
import numpy as np

class module_database:
  def __init__(self, master, status):
    for widget in master.winfo_children():
      widget.destroy()
    self.master = master
    status.set('Database')

    # load database
    '''---Master Path-----------------------------------------------------'''
    master_path = os.path.dirname(__file__)
    master_path = master_path[0:master_path.find("\\" + "Python" + "\\") + 8] + 'database' + '\\'
    database_master_file = master_path + 'database.csv'

    with open(database_master_file) as csvfile:
      database_master_csv = csv.reader(csvfile, delimiter=',')
      database_ID = []
      database_TYPE = []
      database_NAME = []
      database_DATE = []
      database_RAWDATA = []
      database_RESULTS = []
      for row in database_master_csv:
        if len(row) > 0:
          database_ID.append(row[0])
          database_TYPE.append(row[1])
          database_NAME.append(row[2])
          database_DATE.append(row[3])
          database_RAWDATA.append(row[4])
          database_RESULTS.append(row[5])
    # create table
    table_header = ['ID', 'TYPE', 'NAME', 'DATE', 'RAWDATA']
    column_widths = (150, 150, 300, 200, 300)
    self.tree = ttk.Treeview(master = master, columns=table_header, show="headings", height=20)
    self.tree.grid(column=0, row=1, sticky='nsew', in_=master)
    master.grid_columnconfigure(0, weight=1)
    master.grid_rowconfigure(0, weight=1)
    # set headings
    for ix in enumerate(table_header):
      self.tree.heading(table_header[ix[0]], text=table_header[ix[0]], anchor='w')
      self.tree.column(table_header[ix[0]], width=column_widths[ix[0]])
    # set items
    for i in range(1,len(database_NAME)):
      item = [database_ID[i], database_TYPE[i], database_NAME[i], database_DATE[i], database_RAWDATA[i]]
      self.tree.insert('', 'end', values=item)

    # open data => double click
    self.tree.bind("<Double-1>", self.OnDoubleClick)
    self.tree.bind("<Delete>", self.OnDelete)
    # create edit frame
    self.edit_frame = tkinter.Frame(master, width=1100, height=200, bg='#333333')
    self.edit_frame.grid(row=0, column=0)

  def OnDelete(self, event):
    item = self.tree.selection()[0]
    # read item
    item_dict = self.tree.item(item)
    item = item_dict.get('values')
    # delete Raw data
    file_name = item[4]
    os.remove(file_name)
    # delte Results file
    file_name = file_name.replace('_raw.csv','_results.csv')
    os.remove(file_name)
    # delte entry from database
    '''---Master Path-----------------------------------------------------'''
    master_path = os.path.dirname(__file__)
    master_path = master_path[0:master_path.find("\\" + "Python" + "\\") + 8] + 'database' + '\\'
    database_master_file = master_path + 'database.csv'
    with open(database_master_file) as csvfile:
      database_master_csv = csv.reader(csvfile, delimiter=',')
      database_ID = []
      database_TYPE = []
      database_NAME = []
      database_DATE = []
      database_RAWDATA = []
      database_RESULTS = []
      for row in database_master_csv:
        if len(row) > 0:
          if row[0] != str(item[0]):
            database_ID.append(row[0])
            database_TYPE.append(row[1])
            database_NAME.append(row[2])
            database_DATE.append(row[3])
            database_RAWDATA.append(row[4])
            database_RESULTS.append(row[5])
    with open(database_master_file, 'w') as csvfile:
      database_master_csv = csv.writer(csvfile, delimiter=',')
      for row in range(0,len(database_NAME)):
        database_master_csv.writerow([database_ID[row], database_TYPE[row], database_NAME[row], database_DATE[row], database_RAWDATA[row], database_RESULTS[row]])
    self.__init__(self.master)

  def OnDoubleClick(self, event):
    for widget in self.edit_frame.winfo_children():
      widget.destroy()
    item = self.tree.selection()[0]
    # read item
    item_dict = self.tree.item(item)
    item = item_dict.get('values')
    # load raw data
    with open(item[-1]) as csvfile:
      rawdata = csv.reader(csvfile, delimiter=',')
      data = []
      for row in rawdata:
        data.append(row)

    # load fitted data
    fit_item = item[-1][:-7] + 'fit.csv'
    with open(fit_item) as csvfile:
      fitdata = csv.reader(csvfile, delimiter=',')
      fdata = []
      for row in fitdata:
        fdata.append(row)
    # get frame size
    width = 1100
    height = 200

    f = Figure(figsize=(width/100,height/100),
          dpi=100, facecolor='#333333')

    a = f.add_subplot(111,facecolor='#333333')
    for child in a.get_children():
      if isinstance(child, matplotlib.spines.Spine):
        child.set_color('#bbbbbb')
    a.set_xlabel('wavenumber [cm-1]',fontweight='bold', fontsize=12)
    a.set_ylabel('abs.',fontweight='bold', fontsize=12)
    a.xaxis.label.set_color('#bbbbbb')
    a.yaxis.label.set_color('#bbbbbb')
    a.tick_params(axis='x', colors='#bbbbbb')
    a.tick_params(axis='both', which='both', bottom=False, top=False, labelbottom=False, right=False, left=False, labelleft=False)
    a.invert_xaxis()
    x = data[0]
    y = data[1]
    x = list(np.float_(x))
    y = list(np.float_(y))

    fx = fdata[0]
    fy = fdata[1]
    fb = fdata[3]
    fx = list(np.float_(fx))
    fy = list(np.float_(fy) + np.float_(fb))
    fb = list(np.float_(fb))
    a.plot(x,y,lw=2,color='#666666')
    a.plot(fx,fy,lw=2,color='#EE3135')
    f.set_tight_layout(True)
    canvas = FigureCanvasTkAgg(f, self.edit_frame)
    canvas.draw()

    canvas.get_tk_widget().pack(side=tkinter.BOTTOM, fill=tkinter.BOTH, expand=True)
    canvas._tkcanvas.pack(side=tkinter.TOP, fill=tkinter.BOTH, expand=True)
```

Source Code of modified *µIDENT* Library Search Module

Listing 6: python source code

```
"""
@author: Gerrit Renner
date: 2019-02-06

title: library search module
"""

# LOAD PACKAGES
# external
import csv
import tkinter
import numpy as np
np.seterr(divide='ignore', invalid='ignore')
from tkinter.filedialog import askopenfilenames, asksaveasfilename

class module_libsearch:
  def __init__(self, master, status, main):
    for widget in master.winfo_children():
      widget.destroy()
    status.set('Library Search')
    # create buttons
    self.load_references_button_text = tkinter.StringVar()
    self.load_references_button_text.set('Load References')
    self.load_references_button = tkinter.Button(master, textvariable=self.load_references_button_text,
                    font=(None, 15, 'bold'), width=55,
                    relief='groove', bg='#e22b0b',
                    fg='#dddddd', command= lambda: self.load_references(self.load_references_button_text))

    self.load_samples_button_text = tkinter.StringVar()
    self.load_samples_button_text.set('Load Samples')
    self.load_samples_button = tkinter.Button(master, textvariable=self.load_samples_button_text,
                    font=(None, 15, 'bold'), width=55,
                    relief='groove', bg='#e22b0b',
                    fg='#dddddd', command= lambda: self.load_samples(self.load_samples_button_text))

    self.run_button_text = tkinter.StringVar()
    self.run_button_text.set('Run Search')
    self.run_button = tkinter.Button(master, textvariable=self.run_button_text,
                    font=(None, 15, 'bold'), width=55,
                    relief='groove', bg='#bab6b6',
                    fg='#dddddd', command= lambda: self.run_search())

    self.load_references_button.grid(row=0, column=0, sticky='n')
    self.load_samples_button.grid(row=1, column=0, sticky='n')
    self.run_button.grid(row=2, column=0, sticky='n',pady=20)

  def load_references(self,button):
    self.ref_files = askopenfilenames(filetypes=(("Reference files", "*_REF_results.csv"),))
    button.set(str(len(self.ref_files)) + ' Reference(s) Loaded')
    self.load_references_button.configure(background='#317256')
    try:
      if len(self.ref_files) * len(self.smp_files) > 0:
        self.run_button.configure(background='#317256')
    except:
      pass
    self.ref_names = ["samples \ references: "]
    for i in range(len(self.ref_files)):
      _idx = self.ref_files[i].rfind('/')
      _idx2 = self.ref_files[i].rfind('_REF_')
      self.ref_names.append(self.ref_files[i][_idx+1:_idx2])


  def load_samples(self,button):
    self.smp_files = askopenfilenames(filetypes=(("Sample files", "*_SMP_results.csv"),))
    button.set(str(len(self.smp_files)) + ' Samples Loaded')
    self.load_samples_button.configure(background='#317256')
    try:
      if len(self.ref_files) * len(self.smp_files) > 0:
        self.run_button.configure(background='#317256')
    except:
      pass
    self.smp_names = []
    for i in range(len(self.smp_files)):
      _idx = self.smp_files[i].rfind('/')
      _idx2 = self.smp_files[i].rfind('_SMP_')
      self.smp_names.append(self.smp_files[i][_idx+1:_idx2])

  def run_search(self):
    try:
      if len(self.ref_files) * len(self.smp_files) > 0:
        HQI = np.zeros((len(self.smp_files),len(self.ref_files)))
        for j in range(len(self.smp_files)):
          # load sample file
          smp_position, smp_area, smp_significance = self.load_result_file(self.smp_files[j])
          for i in range(len(self.ref_files)):
            # load reference file
            ref_position, ref_area, ref_significance = self.load_result_file(self.ref_files[i])
            hqi = int((1 - self.compare_spectra(ref_position, ref_area, ref_significance, smp_position, smp_area, smp_significance)) * 1000)
            HQI[j,i] = hqi
        self.file_name = asksaveasfilename(filetypes=(("HQI Results", "*.csv"),))
        if self.file_name.find('.csv') == -1:
          self.file_name = self.file_name + '.csv'
        with open(self.file_name, 'w', newline='') as csvfile:
            exportdata = csv.writer(csvfile, delimiter=',')
            exportdata.writerow(self.ref_names)
            #exportdata.writerow(self.smp_names)
            for i in range(len(self.smp_files)):
              row = [self.smp_names[i]]
              for j in range(len(self.ref_files)):
                row.append(int(HQI[i,j]))
              exportdata.writerow(row)
    except:
      pass

  def load_result_file(self, fname):
    with open(fname) as csvfile:
      result_file = csv.reader(csvfile, delimiter=',')
      data = []
      for row in result_file:
        data.append(row)
    position = data[0]
    area = data[1]
    significance = data[2]
    position = list(np.float_(position))
    area = list(np.float_(area))
    significance = list(np.float_(significance))
    position = np.array(position)
    area = np.array(area)
    significance = np.array(significance)
    return position, area, significance

  def compare_spectra(self, ref_position, ref_area, ref_significance, smp_position, smp_area, smp_significance):
    # find similar peaks
    threshold = 10
    flag = True
    posi_ = np.array([])
    area_ = np.array([])
    sign_ = np.array([])
    ref_significance_total = np.sum(ref_significance)
    while flag:
      dist_x = np.zeros([len(ref_position), len(smp_position)])
      for i in range(len(ref_position)):
        for j in range(len(smp_position)):
          dist_x[i,j] = abs(ref_position[i] - smp_position[j])
      idx = np.argmin(dist_x)
      min_min = np.min(dist_x)
      row = int(idx/len(smp_position))
      col = int(idx%len(smp_position))
      posi_ = np.append(posi_,[ref_position[row],smp_position[col]])
      area_ = np.append(area_,[ref_area[row],smp_area[col]])
      sign_ = np.append(sign_,[ref_significance[row],smp_significance[col]])
      ref_position = np.delete(ref_position,row)
      ref_area = np.delete(ref_area,row)
      ref_significance = np.delete(ref_significance,row)
      smp_position = np.delete(smp_position,col)
      smp_area = np.delete(smp_area,col)
      smp_significance = np.delete(smp_significance,col)
      if len(ref_position) * len(smp_position) == 0:
        flag = False
      if min_min > threshold:
        flag = False
        posi_ = posi_[0:-2]
        area_ = area_[0:-2]
        sign_ = sign_[0:-2]

    posi_ = np.reshape(posi_,[int(len(posi_)/2),2])
    area_ = np.reshape(area_,[int(len(area_)/2),2])
    sign_ = np.reshape(sign_,[int(len(sign_)/2),2])
    ref_significance_used = np.sum(sign_[:,0])
    y_ = np.array([])
    s_ = np.array([])
    x_ = np.array([])
    for i in range(len(posi_)-1):
      for j in range(i+1,len(posi_)):
        y_ = np.append(y_, [(area_[i,0]-area_[j,0])/(area_[i,0]+area_[j,0]), (area_[i,1]-area_[j,1])/(area_[i,1]+area_[j,1])])
        s_ = np.append(s_, np.min([sign_[i,0], sign_[j,0], sign_[i,1], sign_[j,1]]))
        x_ = np.append(x_, ((posi_[i,0]-posi_[i,1])**2 + (posi_[j,0]-posi_[j,1])**2)**0.5)
    y_ = np.reshape(y_,[int(len(y_)/2),2])
    dy = ((y_[:,0]-y_[:,1])**2)**0.5
    dx = x_
    k1 = 1.2 # Empirical factors
    k2 = 15 # Empirical factors
    k3 = 4.7 # Empirical factors
    k4 = 5.3 # Empirical factors
    dz = 1 - 1/((np.exp(k1*dx-k2)+1)*(np.exp(k3*dy-k4)+1))
    d = np.sum(dz*s_/np.sum(s_))**(ref_significance_used/ref_significance_total)
    if d == 0:
      d = 1
    return d
```

Source Code of modified *μIDENT* Function Package

Listing 7: python source code

```python
"""
@author: Gerrit Renner
date: 2019-02-06

title: internal functions
"""

# LOAD PACKAGES
# external
import numpy as np
from scipy import interpolate, stats
from scipy import special
from scipy.signal import savgol_filter, argrelextrema
import matplotlib.pyplot as plt
from scipy.sparse import identity,spdiags
from scipy.sparse import csc_matrix, eye, diags
from scipy.sparse.linalg import spsolve
import pywt
from statsmodels.robust import mad
from scipy.optimize import curve_fit
import csv
import statsmodels.api as sm
import tkinter
import matplotlib
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2TkAgg
from matplotlib.figure import Figure
import time
from scipy.interpolate import interp1d
from scipy.interpolate import UnivariateSpline

# Gaussian Function ############################################################
def gaussian_func(x,parameter):
  x0 = parameter[0::3]
  h = parameter[1::3]
  w = parameter[2::3]
  x = np.repeat(np.transpose(np.matrix(x)), len(x0), axis = 1)
  # Function
  x = x - x0
  gaussian = np.multiply(np.exp(np.square(np.divide(x,w)) * - 0.5), h)
  return np.squeeze(np.asarray(np.sum(gaussian, axis = 1)))

# Asymmetric Pseudo Voigt Function ##############################################
def apv_func(x, parameter):
  x0 = parameter[0::5] # Peak Center
  h = parameter[1::5] # Peak Height
  w = parameter[2::5] # Peak Width
  f = parameter[3::5] # Peak Shape Factor
  a = parameter[4::5] # Peak Asymmetry Factor
  x = np.repeat(np.transpose(np.matrix(x)), len(a), axis = 1) # Create X-Matrix
  # Function ------------------------------------------------------------------
  x = x - x0 # Consider Peak Centers
  # Asymmetric Function
  w = np.divide(np.multiply(w,2), 1 + np.exp(np.multiply(x, a)))
  # Gaussian
  gaussian = np.multiply(np.exp(np.square(np.divide(x,w)) * - 0.5), h)
  # Lorentzian
  lorentzian = np.divide(h, np.square(np.divide(x,w)) / 3 + 1)
  # Voigt
  voigt = np.sum(np.multiply(lorentzian, f) + np.multiply(np.subtract(1, f), gaussian),axis = 1)
  return np.squeeze(np.asarray(voigt))

def apv_fit(x, *args):
  x0 = args[0::5] # Peak Center
  h = args[1::5] # Peak Height
  w = args[2::5] # Peak Width
  f = args[3::5] # Peak Shape Factor
  a = args[4::5] # Peak Asymmetry Factor
  x = np.repeat(np.transpose(np.matrix(x)), len(a), axis = 1) # Create X-Matrix
  # Function ------------------------------------------------------------------
  x = x - x0 # Consider Peak Centers
  # Asymmetric Function
  w = np.divide(np.multiply(w,2), 1 + np.exp(np.multiply(x, a)))
  # Gaussian
  gaussian = np.multiply(np.exp(np.square(np.divide(x,w)) * - 0.5), h)
  # Lorentzian
  lorentzian = np.divide(h, np.square(np.divide(x,w)) / 3 + 1)
  # Voigt
  voigt = np.sum(np.multiply(lorentzian, f) + np.multiply(np.subtract(1, f), gaussian), axis = 1)
  return np.squeeze(np.asarray(voigt))

# Whittaker Smooth ##############################################################
def WhittakerSmooth(x,w,lambda_,differences=1):
  '''
  This function is a translation in python of the R source code of airPLS version 2.0 by Yizeng Liang and Zhang Zhimin - https://code.google.com/p/airpls
  Reference: Z.-M. Zhang, S. Chen, and Y.-Z. Liang, Baseline correction using adaptive iteratively reweighted penalized least squares. Analyst 135 (5), 1138-1146 (2010).
  '''
  '''
  Penalized least squares algorithm for background fitting

  input
    x: input data (i.e. chromatogram of spectrum)
    w: binary masks (value of the mask is zero if a point belongs to peaks and one otherwise)
    lambda_: parameter that can be adjusted by user. The larger lambda is, the smoother the resulting background
    differences: integer indicating the order of the difference of penalties

  output
    the fitted background vector
  '''
  X=np.matrix(x)
  m=X.size
  i=np.arange(0,m)
  E=eye(m,format='csc')
  D=E[1:]-E[:-1] # numpy.diff() does not work with sparse matrix. This is a workaround.
  W=diags(w,0,shape=(m,m))
  A=csc_matrix(W+(lambda_*D.T*D))
  B=csc_matrix(W*X.T)
  background=spsolve(A,B)
  return np.array(background)

# airPLS ########################################################################
def airPLS(x, lambda_=100, porder=2, itermax=15):
  '''
  This function is a translation in python of the R source code of airPLS version 2.0 by Yizeng Liang and Zhang Zhimin - https://code.google.com/p/airpls
  Reference: Z.-M. Zhang, S. Chen, and Y.-Z. Liang, Baseline correction using adaptive iteratively reweighted penalized least squares. Analyst 135 (5), 1138-1146 (2010).
  '''
  '''
  Adaptive iteratively reweighted penalized least squares for baseline fitting

  input
    x: input data (i.e. chromatogram of spectrum)
    lambda_: parameter that can be adjusted by user. The larger lambda is, the smoother the resulting background, z
    porder: adaptive iteratively reweighted penalized least squares for baseline fitting

  output
    the fitted background vector
  '''
  m=x.shape[0]
  w=np.ones(m)
  for i in range(1,itermax+1):
    z=WhittakerSmooth(x,w,lambda_, porder)
    d=x-z
    dssn=np.abs(d[d<0].sum())
    if(dssn<0.001*(abs(x)).sum() or i==itermax):
      if(i==itermax):
        print('WARING max iteration reached!')
      break
    w[d>=0]=0 # d>0 means that this point is part of a peak, so its weight is set to 0 in order to ignore it
    w[d<0]=np.exp(i*np.abs(d[d<0])/dssn)
    w[0]=np.exp(i*(d[d<0]).max()/dssn)
    w[-1]=w[0]
  return z

# Check for Significant Peak Prominence ########################################
def peak_combination(x,parameter,baseline):
  # number of peaks
  n_peaks = int(len(parameter)/5)
  index_vector = np.linspace(0,n_peaks-1,n_peaks,dtype=int)
  # peak positions
  peak_positions = parameter[0::5]
  peak_heights = parameter[1::5]
  positions_index = np.zeros(n_peaks,dtype=int)
  for i in range(0,len(peak_positions)):
    positions_index[i] = np.argmin(abs(peak_positions[i] - x))
  # fitted spectrum
  y_fitted = apv_func(x,parameter) + baseline
  # check for peak positions that occure at local minimum
  _filter = (y_fitted[positions_index] < y_fitted[positions_index+1]) * (y_fitted[positions_index] < y_fitted[positions_index-1])
  positions_index[_filter] = positions_index[_filter] + 1

  # combine peaks with identical position
  n_unique_positions = len(np.unique(positions_index))
  unique_peaks = [None] * n_unique_positions
  unique_positions_index = np.unique(positions_index)
  for i in range(n_unique_positions):
    unique_peaks[i] = positions_index[positions_index == unique_positions_index[i]]
  # combine left sided peaks with no turning point
  _not_empty = [None]*n_unique_positions
  for i in range(n_unique_positions-1):
    # left sided peak
    actual_main_peak = max(unique_peaks[i])
    # right sided neighbour
    neighbour_peak = min(unique_peaks[i+1]+1)
    # tmp dataset
    _tmp_data = y_fitted[actual_main_peak:neighbour_peak]
    _local_minima = argrelextrema(_tmp_data, np.less)
    # test for minimum and combine peaks if needed
    if len(_local_minima[0]) == 0: # np.argmin(_tmp_data) == 0:
      # add actual peaks to new array
      unique_peaks[i+1] = np.append(unique_peaks[i+1],unique_peaks[i])
      # delete actual peaks from old array
      unique_peaks[i] = [None]
    _not_empty[i] = type(unique_peaks[i]) is np.ndarray
  _not_empty[-1] = type(unique_peaks[-1]) is np.ndarray

  # delete empty peak arrays
  unique_peaks2 = [None] * sum(_not_empty)
  j = 0
  for i in range(n_unique_positions):
    if _not_empty[i]:
      unique_peaks2[j] = unique_peaks[i]
      j = j+1
  unique_peaks = unique_peaks2

  # combine right sided peaks with no turning point
  _not_empty = [None] * len(unique_peaks)
  for i in range(len(unique_peaks)-1):
    j = len(unique_peaks) - i -1
    # right sided peak
    actual_main_peak = min(unique_peaks[j]+1)
    # left sided neighbour
    neighbour_peak = max(unique_peaks[j-1])
    # tmp dataset
    _tmp_data = y_fitted[neighbour_peak:actual_main_peak]
    _local_minima = argrelextrema(_tmp_data, np.less)
    # test for minimum and combine peaks if needed
    if len(_local_minima[0]) == 0:
      # add actual peaks to new array
      unique_peaks[j-1] = np.append(unique_peaks[j-1],unique_peaks[j])
      # delete actual peaks from old array
      unique_peaks[j] = [None]
    _not_empty[j] = type(unique_peaks[j]) is np.ndarray
  _not_empty[0] = type(unique_peaks[0]) is np.ndarray

  # delete empty peak arrays
  unique_peaks2 = [None] * sum(_not_empty)
  j = 0
  for i in range(len(unique_peaks)):
    if _not_empty[i]:
      unique_peaks2[j] = unique_peaks[i]
      j = j+1
  unique_peaks = unique_peaks2
  # define index vector
  index_vector2 = []
  for i in range(len(unique_peaks)):
    _index = []
    for j in range(len(unique_peaks[i])):
      actual_index = index_vector[unique_peaks[i][j] == positions_index]
      _index = np.append(_index,actual_index)
    _index = np.unique(_index)
    index_vector2 = np.append(index_vector2,_index * 0 + _index[0])

  return index_vector2

# Initialize Peaks (MASTER FUNCTION) ###########################################
def peak_init(x,y,y_raw):
  # x:    x Data
  # y:    y Data (smoothed + baseline corrected)
  # y_raw: y Data (baseline corrected)

  # Preset Fit Values
  pre_y = np.array([])
  parameter = np.array([])
  peak_x = np.array([])
  peak_y = np.array([])
  pre_residues = y
  rel_residues = 1
  k = -1
  while rel_residues > 0.05:
    k = k + 1
    if k == 0:
      pre_residues = savgol_filter(pre_residues,11,2)
    # Main Peaks
    main_x, main_y, main_p, filt = peakProminence(x, pre_residues)
    peak_x = np.append(peak_x,main_x[filt])
    peak_y = np.append(peak_y,main_y[filt])
    peak_x, peak_y = correctPosition(x,y_raw,peak_x)
    peak_x, indices = np.unique(peak_x,return_index=True)
    peak_y = peak_y[indices]
    charc_x, charc_y = characteristicPoints(x,y_raw,peak_x,peak_y,'right')
    charc_x2, charc_y2 = characteristicPoints(x,y_raw,peak_x,peak_y,'left')
    pre_widths = (charc_x - charc_x2)/2
    # check if peak_x and charc_x are equal
    peak_x = peak_x[pre_widths > 0]
    peak_y = peak_y[pre_widths > 0]
    pre_widths = pre_widths[pre_widths > 0]
    # Filter Water Bands
    min_width = 1.5
    min_wavenumber = 1800
    _filter = (peak_x > min_wavenumber) * (pre_widths < min_width)
    peak_x = peak_x[~_filter]
    peak_y = peak_y[~_filter]
    pre_widths = pre_widths[~_filter]
    # Delete Double Bands
    _filter = np.array([True],dtype=bool)
    for band in peak_x:
      # check distances
      _distances = np.sort(abs(peak_x - band))
      if (_distances[1] < 2) * (_filter[-1]):
        _filter = np.append(_filter,[False])#False
      else:
        _filter = np.append(_filter,[True])
    _filter = _filter[1:]
    peak_x = peak_x[_filter]
    peak_y = peak_y[_filter]
    pre_widths = pre_widths[_filter]
    # Filter Bands with Width or Height = 0
    _filter = (peak_y < np.std(np.diff(y_raw))*3) + (pre_widths == 0)
    peak_x = peak_x[~_filter]
    peak_y = peak_y[~_filter]
    pre_widths = pre_widths[~_filter]
    # Set Parameters
    parameter = np.zeros(len(peak_x)*3)
    parameter[0::3] = peak_x
    parameter[1::3] = peak_y
    parameter[2::3] = pre_widths
    # Correct heights of overlayed peaks
    # peaks*heights = y
    peaks = []
    for i in range(len(peak_x)):
      peaks.append(np.exp( -.5 * ((peak_x[i] - peak_x) / pre_widths)**2))
    try:
      heights = np.linalg.solve(peaks,peak_y)
      print('peak initialize optimal')
    except:
      heights = peak_y
      print('peak initialize not optimal')
    parameter[1::3] = heights
    # Filter negative heights
    _filter = np.repeat(parameter[1::3] > 0, 3)
    parameter = parameter[_filter]
    # Parameter for residues calculation
    parameter_estimated = np.zeros(len(parameter))
    parameter_estimated[0::3] = parameter[0::3]
    parameter_estimated[1::3] = parameter[1::3]#*1.2
    parameter_estimated[2::3] = parameter[2::3]#*1.2
    # Calculate residues
    y_estimated = gaussian_func(x,parameter_estimated)
    pre_residues = y - y_estimated
    pre_residues[pre_residues < 0] = 0
    rel_residues = sum(pre_residues) / sum(y)
    parameter_fitted = parameter
    if k == 3:
      rel_residues = 0
  # Filter Water Spikes
  pos = parameter_fitted[0::3]
  heights = parameter_fitted[1::3]
  widths = parameter_fitted[2::3]
  _filter = (pos > min_wavenumber) * (widths < min_width)
  _filter = _filter + (pos > min_wavenumber) * (heights < 0.05)
  pos = pos[~_filter]
  heights = heights[~_filter]
  widths = widths[~_filter]
  # Set Parameters
  results_parameter = np.zeros(len(pos)*5)
  results_parameter[0::5] = pos
  results_parameter[1::5] = heights
  results_parameter[2::5] = widths
  results_parameter[3::5] = 0
  results_parameter[4::5] = 0
  return results_parameter


# Is Number Function ############################################################
def is_number(x):
  try:
    float(x)
    return True
  except ValueError:
    return False

# Wavelet Denoise ###############################################################
def waveletSmooth( x, wavelet="db4", level=1, title=None ):
  '''
  This function was created by Connor Johnson - http://connor-johnson.com/2016/01/24/using-pywavelets-to-remove-high-frequency-noise/
  '''
  # calculate the wavelet coefficients
  coeff = pywt.wavedec( x, wavelet, mode="per" )
  # calculate a threshold
  sigma = mad( coeff[-level] )
  # changing this threshold also changes the behavior,
  # but I have not played with this very much
  uthresh = sigma * np.sqrt( 2*np.log( len( x ) ) )
  coeff[1:] = ( pywt.threshold( i, value=uthresh, mode="soft" ) for i in coeff[1:] )
  # reconstruct the signal using the thresholded coefficients
  y = pywt.waverec( coeff, wavelet, mode="per" )
  return y

# Find Peaks with Prominence ####################################################
def peakProminence(x,y):
  # find all local maxima in a window of n = 5 datapoints
  n = 5
  b1 = (y[(n-5):-(n-1)] < y[(n-4):-(n-2)])
  b2 = (y[(n-4):-(n-2)] < y[(n-3):-(n-3)])
  b3 = (y[(n-3):-(n-3)] > y[(n-2):-(n-4)])
  b4 = (y[(n-2):-(n-4)] > y[(n-1):])

  B1 = b1
  B2 = ~b2
  B3 = ~b3
  B4 = b4
  B5 = y[(n-3):-(n-3)] > y[(n-5):-(n-1)]
  B6 = y[(n-3):-(n-3)] > y[(n-1):]

  peak_index = np.append([False, False],b1 * b2 * b3 * b4 + B1 * B2 * B3 * B4 * B5 * B6)
  peak_index = np.append(peak_index, [False, False])

  if len(x) == len(peak_index):
    peak_x = x[peak_index]
    peak_y = y[peak_index]
  else:
    peak_x = x[peak_index[0:-1]]
    peak_y = y[peak_index[0:-1]]
  peak_prominence = np.zeros(len(peak_x))

  for i in range(0,len(peak_x)):
    # current peak
    curr_peak_x = peak_x[i]
    curr_peak_y = peak_y[i]
    # peaks that are higher than current peak
    if curr_peak_y < max(peak_y):
      higher_peaks_x = peak_x[peak_y > curr_peak_y]
      # closest higher peak
      idx = np.argmin(abs(curr_peak_x - higher_peaks_x))
      next_peak_x = higher_peaks_x[idx]
      if next_peak_x != curr_peak_x:
        # calculate prominence
        tmp_start = min([curr_peak_x , next_peak_x])
        tmp_stop = max([curr_peak_x , next_peak_x])
        tmp_y = y[(x > tmp_start) * (x < tmp_stop)]
        peak_prominence[i] = curr_peak_y - min(tmp_y)
    else:
      peak_prominence[i] = curr_peak_y - min(y)

  # mean diff y
  mean_diff_y = np.mean(abs(np.diff(y)))
  std_diff_y = np.std(abs(np.diff(y)))
  _filter = peak_prominence > mean_diff_y + 2 * std_diff_y

  return peak_x, peak_y, peak_prominence, _filter


# Correct Peak Positions ########################################################
def correctPosition(x,y,peak_x):
  corr_x = np.zeros(len(peak_x))
  corr_y = np.zeros(len(peak_x))
  for j in range(len(peak_x)):
    i = np.argmin(abs(x - peak_x[j]))
    actual_pos = x[i]
    actual_y = y[i]
    if (i > 1) * (i<len(x)-3):
      actual_test_range = y[i-2:i+2]
      if max(actual_test_range) == actual_y:
        flag = False
      else:
        flag = True
    else:
      flag = False
    while flag:
      i = np.argmax(actual_test_range) - 2 + i
      actual_pos = x[i]
      actual_y = y[i]
      if (i > 1) * (i<len(x)-3):
        actual_test_range = y[i-2:i+2]
        if max(actual_test_range) == actual_y:
          flag = False
        else:
          flag = True
      else:
        flag = False
    corr_x[j] = actual_pos
    corr_y[j] = actual_y
  return corr_x,corr_y


# Find Characteristic Points ####################################################
def characteristicPoints(x,y,peak_x,peak_y,direction='right'):
  charc_x = np.zeros(len(peak_x))
  charc_y = np.zeros(len(peak_x))
  if direction == 'right':
    if len(peak_x) > 0:
      for i in range(len(peak_x)-1):
        x_range = x[(x >= peak_x[i]) * (x <= peak_x[i+1])]
        y_range = y[(x >= peak_x[i]) * (x <= peak_x[i+1])]
        fwhm_y = y_range[y_range <= peak_y[i]/2]
        fwhm_x = x_range[y_range <= peak_y[i]/2]
        try:
          fwhm_y = fwhm_y[0]
          fwhm_x = fwhm_x[0]
        except:
          fwhm_y = min(y_range)
          fwhm_x = x_range[np.argmin(y_range)]
        charc_x[i] = fwhm_x
        charc_y[i] = fwhm_y


      i = len(peak_x) - 1
      x_range = x[x >= peak_x[i]]
      y_range = y[x >= peak_x[i]]
      fwhm_y = y_range[y_range <= peak_y[i]/2]
      fwhm_x = x_range[y_range <= peak_y[i]/2]
      try:
        fwhm_y = fwhm_y[0]
        fwhm_x = fwhm_x[0]
      except:
        fwhm_y = min(y_range)
        fwhm_x = x_range[np.argmin(y_range)]
      charc_x[i] = fwhm_x
      charc_y[i] = fwhm_y

  if direction == 'left':
    if len(peak_x) > 0:
      for i in range(len(peak_x)):
        if i == 0:
          charc_x[i] = 0
          charc_y[i] = 0
        else:
          x_range = x[(x <= peak_x[i]) * (x >= peak_x[i-1])]
          y_range = y[(x <= peak_x[i]) * (x >= peak_x[i-1])]
          fwhm_y = y_range[y_range <= peak_y[i]/2]
          fwhm_x = x_range[y_range <= peak_y[i]/2]
          try:
            fwhm_y = fwhm_y[-1]
            fwhm_x = fwhm_x[-1]
          except:
            fwhm_y = min(y_range)
            fwhm_x = x_range[np.argmin(y_range)]
          charc_x[i] = fwhm_x
          charc_y[i] = fwhm_y


      i = 0
      x_range = x[x <= peak_x[i]]
      y_range = y[x <= peak_x[i]]
      fwhm_y = y_range[y_range <= peak_y[i]/2]
      fwhm_x = x_range[y_range <= peak_y[i]/2]
      try:
        fwhm_y = fwhm_y[-1]
        fwhm_x = fwhm_x[-1]
      except:
        fwhm_y = min(y_range)
        fwhm_x = x_range[np.argmin(y_range)]
      charc_x[i] = fwhm_x
      charc_y[i] = fwhm_y


  return charc_x, charc_y

def running_mean(x, N):
  cumsum = np.cumsum(np.insert(x, 0, 0))
  return (cumsum[N:] - cumsum[:-N]) / float(N)


def ir_smooth(x,y, _window = 360, _overlap = 60):
  N = int(np.trunc(len(x) / _overlap))
  Y = np.zeros([len(x),N])
  for i in range(N):
    xtmp = x[i*_overlap:(i+1)*_overlap + (_window - _overlap)]
    ytmp = y[i*_overlap:(i+1)*_overlap + (_window - _overlap)]
    x1,y1,y2,W = variable_smooth(xtmp,ytmp)
    Y[i*_overlap:(i+1)*_overlap + (_window - _overlap),i] = y2

  Y[Y==0] = np.nan
  Ymedian = np.nanmedian(Y,axis=1)
  return Ymedian

def variable_smooth(x,y):
  W = 100
  _window = 11
  while W > 3E-2:
    y2 = savgol_filter(y,_window,2)
    _fft = np.fft.fft(y2)
    _fft = np.sort(_fft.real)
    x1,y1,w = width_fft(_fft,x)
    W = w
    _window = _window + 2
  return x1,y1,y2,W

def width_fft(_fft,x):
  _fft = _fft[:-1]
  _fft,_idx = np.unique(_fft,return_index = True)
  _iqr = stats.iqr(_fft)
  x1 = x[:-1]
  x1 = x[_idx]
  x1 = (x1 - min(x1)) / (max(x1) - min(x1))

  _quart75 = np.percentile(_fft,75)
  _quart25 = np.percentile(_fft,25)
  _idx = (_fft > _quart25 - 1.5 * _iqr) * (_fft < _quart75 + 1.5 * _iqr)
  _fft = _fft[_idx]
  x1 = x1[_idx]

  f = interp1d(_fft,x1)
  x2 = np.linspace(min(_fft),max(_fft),2*len(_fft))
  y2 = f(x2)

  f = UnivariateSpline(x2,y2, k=5)
  y3 = f(x2)
  y3 = savgol_filter(y3,11,2,deriv=1)
  _idx = argrelextrema(y3, np.greater)
  _idx2 = argrelextrema(y3, np.less)
  try:
    w = abs(np.min(np.max(x2[_idx]) - x2[_idx2]))
  except:
    w = 0
  return x2,y3,w

def plot_data(x,y,master):
  # create figure
  f = Figure(figsize=(100,100),
        dpi=100, facecolor='#333333')
  a = f.add_subplot(111,facecolor='#333333')
  for child in a.get_children():
    if isinstance(child, matplotlib.spines.Spine):
      child.set_color('#bbbbbb')
  a.set_xlabel(cfg_dict.get('xunit'),fontweight='bold', fontsize=16)
  a.set_ylabel(cfg_dict.get('yunit'),fontweight='bold', fontsize=16)
  a.xaxis.label.set_color('#bbbbbb')
  a.yaxis.label.set_color('#bbbbbb')
  a.tick_params(axis='x', colors='#bbbbbb')
  a.tick_params(axis='y', colors='#bbbbbb')
  a.invert_xaxis()
  a.plot(x,y,lw=3,color='#EE3135')
  f.set_tight_layout(True)
  canvas = FigureCanvasTkAgg(f, master)
  canvas.draw()
  canvas.get_tk_widget().pack(side=tkinter.BOTTOM, fill=tkinter.BOTH, expand=True)
  canvas._tkcanvas.pack(side=tkinter.TOP, fill=tkinter.BOTH, expand=True)
```

Source Code of modified *µIDENT*
Graphics