Python Raman Analyzer (PyRANA)

The Raman spectral analysis library

(last update: May 19, 2021)

Author:
Sébastien Manigand (sebastien.manigand83@gmail.com)

Centre for Terrestrial and Planetary exploration (C-TAPE), University of Winnipeg
515, Portage Avenue, Winnipeg R3B 2W9 MB, Canada

Contents

---

1 Introduction

The Python Raman Analyzer (PyRANA) is a python library that provides all the necessary functions to analyze a Raman spectrum without any prior knowledge on the composition of the sample or the shape of the continuum. This library is part of the SALSA database and the lunar Raman spectrometer project (LunaR), which is an instrument under development that will be onboard the next lunar rover, funded by the Lunar Exploration Accelerated Program (LEAP) from the Canadian Space Agency (CSA). The library is available on the SALSA database website: XXXX

1.1 Citations

The SALSA database and the analyzer have been published. In addition, the dependencies also have their associated publications. If you make use of this library in an analysis that you plan to publish, we kindly ask you to cite the following paper when you describe your analysis:

- Manigand, S., et al., 2021, JOURNAL TBD, 42, 123, *"The Spectral Analyses of Lunar Soils and Analogues database (SALSA): a Raman spectral database for lunar exploration"*

We also kindly ask you to add this paragraph to the acknowledgements:

This research has made use of community-developed core Python packages for astronomy and scientific computing including Scipy (Jones et al. 2001, Virtanen et al. 2020), Numpy (van der Walt et al. 2011) and Matplotlib (Hunter 2007).

with the following cited papers:

- Jones, E., Oliphant, T. E., Peterson, P., 2001, online, *"SciPy: Open source scientific tools for Python",*
- Virtanen, P., Gommers, R., Oliphant, T. E., 2020, Nature Methods, 17, 261-272, *"SciPy 1.0: fundamental algorithms for scientific computing in Python",*
- van der Walt, S., Colbert, S. C., Varoquaux, G., 2011, Computing in Science and Engineering, 13, 2, 22-30, *"The NumPy Array: A Structure for Efficient Numerical Computation",*
- Hunter, J. D., 2007, Computing in Science and Engineering, 9, 3, 90-95 *"Matplotlib: A 2D Graphics Environment".*

You will find the necessary BibTex entries at this address (XXXX).

2 Installation

This section explains how to install the library and describes the dependencies.

2.1 Dependencies

This library uses the common scientific python packages and a few utilities to plot and read csv files. The dependencies are:

- **NumPy** (vectorized arrays, mathematical functions),
- **SciPy** (fitting algorithm, statistics),
- **Matplotlib** (plotting, figure formatting),
- **csv** (.csv datafile reading/writing).

2.2 How to install

2.2.1 python environment

There are several ways to install a python environment, with or without the dependencies.

The simplest way is to install Anaconda, an all-in-one python development environment including the python core, many packages, an user-friendly interface for setting and extending the libraries, and a fancy notepad-like for python: Spyder. Anaconda conveniently includes all the dependencies by default, therefore you won't have to install them manually.

2.2.2 PyRANA library

The PyRANA library is not compiled, therefore, you will just have to copy the PyRANA python script in the same directory as your script and add the following line in your code:

```
from PyRANA_v1 import *
```

You can test if the package is correctly read by executing the code with this line alone. If the console does not comply, then you are good to go!

3 Using PyRANA

3.1 The PyRANA class-object

The library allows the user to fit a spectrum only writing a few lines of code. After importing the PyRANA library, you have to create a pyrana class-object:

```
pyrana = PyRANA()
```

The `pyrana` object will be used as a black box to execute the different steps of the spectral analysis. The next step is to load your spectrum with `pyrana`.

3.2 Opening data files

There are three file format accepted by PyRANA, so far: the METAFILE type, the SALSA export type, and a standard ASCII type. The file type are described in section XXXX.

The function to use for METAFILE files is:

```
pyrana.loadfile_METAFILE("filename.meta")
```

The function to use for SALSA export files is:

```
pyrana.loadfile_SALSA("filename.csv")
```

The function to use for ASCII export files is:

```
pyrana.loadfile_ASCII("filename.txt")
```

The spectrum is loaded in the `pyrana` object in `X` and `Y` attributes.

3.3 Overview plot

In order to check if the data has been correctly loaded, you can plot the entire spectrum using the command:

```
pyrana.plot_all()
```

The following figure should appear in the console:

3.4 Cleaning the data

The spectrum may have some *hot pixels* which can badly affect the quality of the fit. For example, Figure 1 shows several hot pixels, with one particularly intense at $\sim$1600 cm^{-1} Raman shift (several times the maximum of the Raman signatures). You can clean-up the spectrum of most of these features by using:

```
pyrana.cleanSpectrum()
```

The default use of this function bases the detection of hot pixels on three times the root mean-square (rms). Optional parameters allow the user to define the rms value to use, to ignore some pixel, or to force the correction of some specific pixels. An example of use of all the optional parameters is written below:

```
pyrana.cleanSpectrum(rms = 83.2, ignore = [2327], force = [664, 879])
```

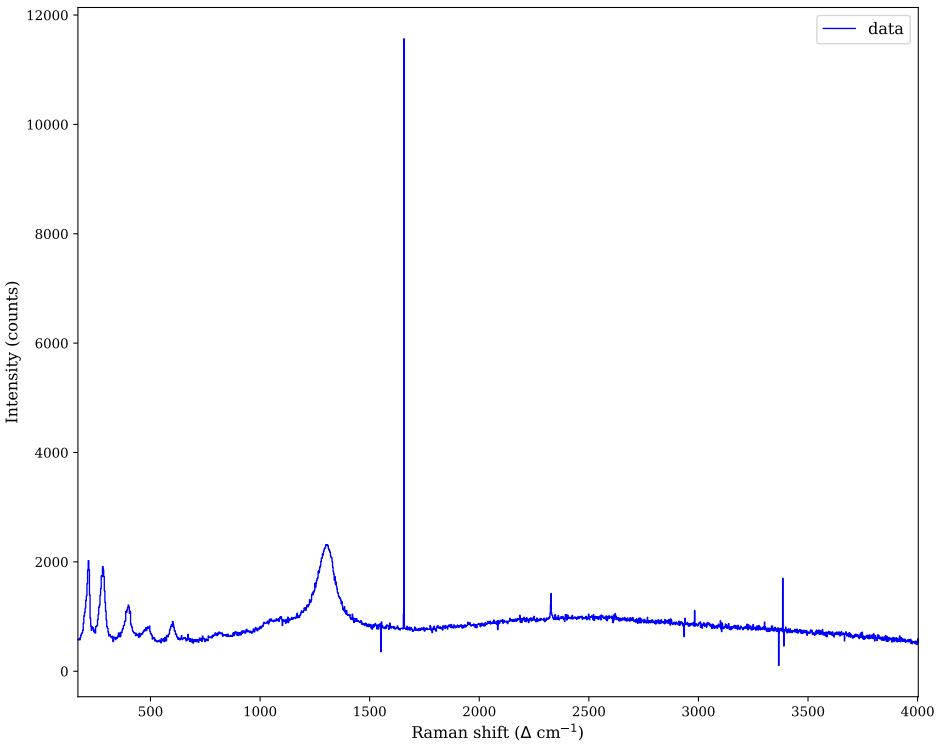

Figure 1: Overview of a loaded Raman spectrum. The data comes from the SALSA database, the sample HEM101, spot 3.

Figure 2 shows an example of spectrum cleaning, before and after the clean.

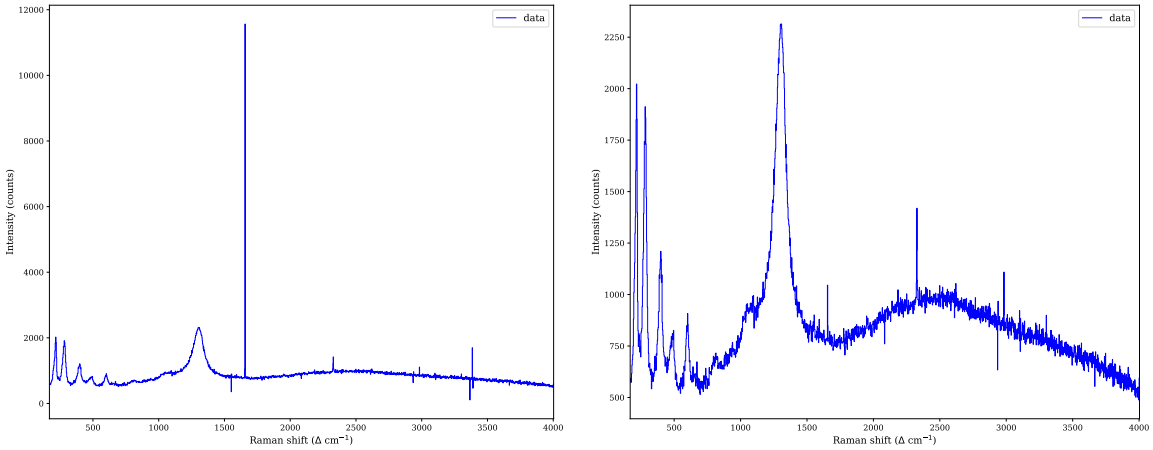

Figure 2: Example of cleaning spectrum. The left-hand and right-hand side spectrum correspond to before and after the cleaning, respectively.

3.5 Fitting the data

You can fit the spectrum by small parts at a time using the function `quickfit`. The function takes a few parameters to define the spectral range to fit, the type of continuum background, and the list of components, up to five. For example, the fit of the most intense peak of Figure 2 cleaned spectrum gives:

```
fit = pyrana.quickfit(Xmin=950, Xmax=1600, continuum='linear', peaks=[
{'profile':  'gauss', 'x0':  1050, 's':20},
{'profile':  'assymCauchy', 'x0':  1300, 's':40} ])
```

You can plot the result of the fit using:

```
pyrana.plot_fit(fit)
```

You can also overlay the fit over the entire spectrum using:

```
pyrana.plot_all(fits=[fit])
```

Figure 3 shows the plot of the fit and the overlay plot.

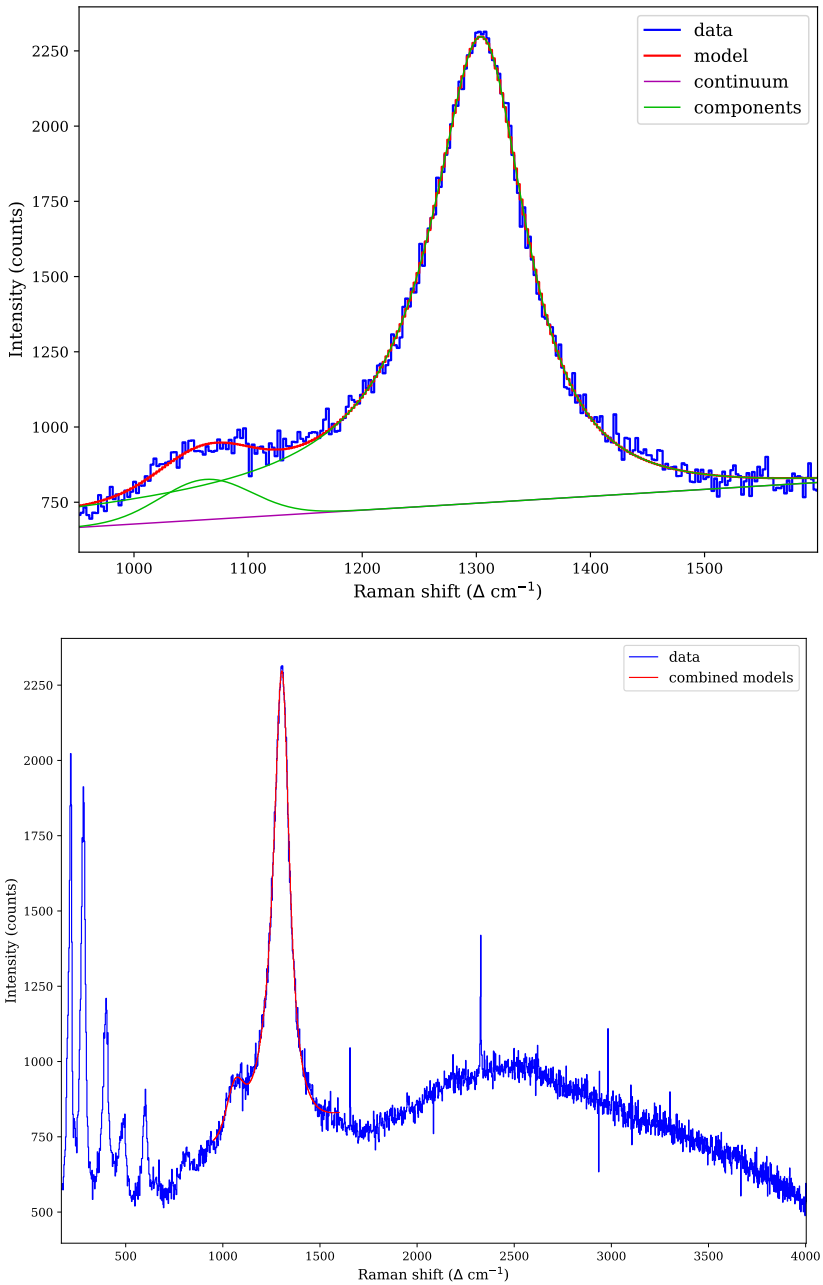

Figure 3: Example of peak fitting and overlay plotting.

3.6 Getting outputs

After fitting all the region of the spectrum you wanted, it is time to get some outputs. In the following, we will consider these fits:

```
fit1 = pyrana.quickfit(Xmin=325, Xmax=550, continuum='linear', peaks=[
'profile':  'cauchy', 'x0':  400,
'profile':  'cauchy', 'x0':  500 ])

fit2 = pyrana.quickfit(Xmin=700, Xmax=950, continuum='poly2', peaks=[
'profile':  'gauss', 'x0':  800, ])

fit3 = pyrana.quickfit(Xmin=950, Xmax=1600, continuum='linear', peaks=[
{'profile':  'gauss', 'x0':  1050, 's':20},
{'profile':  'assymCauchy', 'x0':  1300, 's':40} ])
```

You can get the plot of the fit as a figure in PNG or PDF using:

```
pyrana.plot_fit(fit3, output="fit-result_950-1600.pdf")
```

If you want to use your own plot method, you can export the plot in an ASCII file using:

```
pyrana.plot_fit(fit3, ascii_output="fit-result_950-1600.txt")
```

Once you finished to fit the spectrum, you can resume the fit results, i.e. the parameters of all the peaks, in an ASCII file, using the function `str_peaks` in the example code:

```
f = open("fit_report.txt", 'w')
f.write("##########################################\n")
f.write("# FIT REPORT: {0}\n".format("Final batch 1 and 2"))
f.write("##########################################\n")
f.write("#\n")
f.write("# Profile name\tcenter\terror\tintensity\terror\twidth\terror\t
assymmetry\terror\tchi2\tN\tprob\n")
f.write("#\n# {0}\n".format(pyrana.dataname))
f.write(pyrana.str_peaks('ascii', [fit1, fit2, fit3]))
f.close()
```

Finally, you can generate a log file containing all the fit that has been made during the execution of your script using:

```
pyrana.log()
```

4 List of functions

4.1 assymCauchy

```
PyRANA.assymCauchy(X, I, x0, s, a)
```

The function `assymCauchy` returns the asymmetric Lorentzian function evaluation of `X`, a single value or a `numpy.array`, with `I`, `x0`, and `s` the intensity, the position and the width of the Lorentzian, respectively, and `a` the asymmetry factor:

$$\gamma(X) = \frac{2s}{1 + \exp\left(a\left(X - x_0\right)\right)} \tag{1}$$

$$\text{assymCauchy}(X) = \frac{I}{1 + \left(\frac{X - x_0}{\gamma(X)}\right)^2} \times w_{\text{damp}}(X) \tag{2}$$

with $w_{\text{damp}}(X) = \text{gauss}(X;\ 1,\ x_0,\ 5s)$ a weight function that damps the long range tail of the profile to get a better fit of the continuum with respect to the peak.

The asymmetric profile is detailed in the study of Stancik and Brauns (2008).

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

Y = pyrana.assymCauchy(X, 10, 0, 2, 0.1)
```

4.2 assymGauss

```
PyRANA.assymGauss(X, I, x0, s, a)
```

The function `assymGauss` returns the asymmetric Gaussian function evaluation of `X`, a single value or a `numpy.array`, with `I`, `x0`, and `s` the intensity, the position and the width of the Gaussian, respectively, and `a` the asymmetry factor:

$$\gamma(X) = \frac{2s}{1 + \exp\left(a\left(X - x_0\right)\right)} \tag{3}$$

$$\text{assymGauss}(X) = I\ \exp\frac{-\left(X - x_0\right)^2}{2\gamma(X)^2} \tag{4}$$

The asymmetric profile is detailed in the study of Stancik and Brauns (2008).

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

Y = pyrana.assymGauss(X, 10, 0, 2, 0.1)
```

4.3 bwfGraphene

```
PyRANA.bwfGraphene(X, I, x0, s, a)
```

The function `bwfGraphene` returns the Breit-Wigner-Fano (BWF) profile evaluation of `X`, a single value or a `numpy.array`, with `I`, `x0`, and `s` the intensity, the position and the width of the peak, respectively, and `a` the asymmetry factor:

$$\gamma = \frac{X - x_0}{s} \tag{5}$$

$$\text{bwfGraphene}(X) = I\left[a^2 + \frac{1 + 2a\gamma - a^2}{1 + \gamma^2}\right] \tag{6}$$

The BWF profile describes the specific vibration mode of ordered graphene crystal. It has been determined by Klein (1975) and Eklund and Subbaswamy (1979). An extensive study of the BWF profile can be found in Hasdeo et al. (2014).

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

Y = pyrana.bwfGraphene(X, 10, 0, 2, -0.1)
```

4.4 cauchy

```
PyRANA.cauchy(X, I, x0, s)
```

The function `cauchy` returns the Lorentzian function evaluation of `X`, a single value or a `numpy.array`, with `I`, `x0`, and `s` the intensity, the position and the width, respectively:

$$\text{cauchy}(X) = \frac{I}{1 + \left(\frac{X - x_0}{s}\right)^2} \times w_{\text{damp}}(X) \tag{7}$$

with $w_{\text{damp}}(X) = \text{gauss}(X;\ 1,\ x_0,\ 5s)$ a weight function that damps the long range tail of the profile to get a better fit of the continuum with respect to the peak.

The FWHM of a Lorentzian profile corresponds to $FWHM = 2s$.

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

Y = pyrana.cauchy(X, 10, 0, 2)
```

4.5 chi2_

```
PyRANA.chi2_(self, Y, Yerr, Yfit)
```

The function `chi2_` returns the chi-square of `Y` with the model `Yfit`, given the uncertainties `Yerr`:

$$\text{chi2\_} = \sum \frac{(Y - Y_{\text{fit}})^2}{Y_{\text{err}}^2} \tag{8}$$

If the model has also an uncertainty, please combine it with the data uncertainty as follow: $\texttt{Yerr}^2 = \text{data}_{\text{error}}^2 + \text{model}_{\text{error}}^2$.

Example:

```
pyrana = PyRANA()
pyrana.loadfile_ASCII('filename.txt')
fit = pyrana.quickfit(Xmin=700, Xmax=950, continuum='linear',
peaks=[{'profile':  'gauss', 'x0':  800} ])

chiSquare = pyrana.chi2_(fit['dataY'], fit['dataYerr'], fit['model'])
print("chi-square = {0:.2e}".format(chiSquare))
```

4.6 cleanSpectrum

```
PyRANA.cleanSpectrum(rms=None, comment=False, ignore=None, forced=None)
```

The function `cleanSpectrum` detects the inconsistent pixels, called 'hot pixels', and correct them. A pixel is considered when its difference from the local average is greater than 3`rms`. If no value is given to `rms`, then the function uses the result from `getRMS()`.

You can ignore, or force, the cleaning of particular pixels by listing them in `ignore` and `forced`, respectively. If `comment` is `True`, additional information about the execution of the function are printed out.

Example:

```
pyrana = PyRANA()
pyrana.loadfile_SALSA("salsa-export.csv")

# Default cleaning
pyrana.cleanSpectrum()

# Ignored and Forced example
pyrana.cleanSpectrum(ignore=[1625, 2346], forced=[650])
```

4.7 fctWrapper2

```
PyRANA.fctWrapper2(X, fctdef, a, b)
```

The function `fctWrapper2` returns the 2-parameters function defined by the value of `fctdef`, passing the parameter `X`, `a`, and `b` to the returned function, in the same order. The permitted value of `fctdef` and the corresponding functions are:

- `'poly1'`: returns a `poly1` function.

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

fct = pyrana.fctWrapper2(X, 'poly1', 1, -1)
```

4.8 fctWrapper3

```
PyRANA.fctWrapper3(X, fctdef, a, b, c)
```

The function `fctWrapper3` returns the 3-parameters function defined by the value of `fctdef`, passing the parameter `X`, `a`, `b`, and `c` to the returned function, in the same order. The permitted value of `fctdef` and the corresponding functions are:

- `'poly2'`: returns a `poly2` function,
- `'gauss'`: returns a `gauss` function,
- `'cauchy'`: returns a `cauchy` function.

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

fct = pyrana.fctWrapper3(X, 'gauss', 10, 0.5, 2)
```

4.9 fctWrapper4

```
PyRANA.fctWrapper4(X, fctdef, a, b, c, d)
```

The function `fctWrapper4` returns the 4-parameters function defined by the value of `fctdef`, passing the parameter `X`, `a`, `b`, `c`, and `d` to the returned function, in the same order. The permitted value of `fctdef` and the corresponding functions are:

- `'assymCauchy'`: returns a `assymCauchy` function,
- `'assymGauss'`: returns a `assymGauss` function.
- `'BWFGraphene'`: returns a `bwfGraphene` function.

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

fct = pyrana.fctWrapper4(X, 'assymGauss', 10, 0.5, 2, 0.1)
```

4.10 gauss

```
PyRANA.gauss(X, I, x0, s)
```

The function `gauss` returns the Gaussian function evaluation of `X`, a single value or a `numpy.array`, with `I`, `x0`, and `s` the intensity, the position and the width, respectively:

$$\text{gauss}(X) = I \ \exp \frac{-(X - x_0)^2}{2s^2} \tag{9}$$

The FWHM of a Gaussian profile corresponds to $FWHM = s\sqrt{2\ln 2}$.

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

Y = pyrana.gauss(X, 10, 0, 2)
```

4.11 getRMS

```
PyRANA.getRMS(wn = None, spec = None, method='fft', _DEBUG = False)
```

Returns the Root Mean Square (RMS) of the data, where `wn` and `spec` correspond to the wavenumbers and the spectrum arrays. If nothing is given for `wn` or `spec`, the function uses the `PyRANA.X` and `PyRANA.Y` arrays stored into the class-object, for example, those after loading a data file. The `method` parameter sets the method used to determine the RMS. If `_DEBUG` is `True`, the function returns `distrib_X`, `distrib_Y`, `I`, `s*(numpy.sqrt(2*numpy.log(2)))`, `x0`, corresponding to the X and Y arrays of the filtered distribution, the intensity, the FWHM, and the position of the Gaussian fit of the distribution.

The implemented RMS method are:

- `'fft'`: filters the short spatial frequencies of the Fourier transform of the spectrum and invert the transformation. The long spatial frequencies of the Fourier transform carries the short-ranged frequencies of the spectrum, that is the noise fluctuations. The filtered spectrum contains only the fluctuations. Then, the RMS is given by the FWHM of the distribution of the fluctuations.

Example:

```
pyrana = PyRANA()
pyrana.loadfile_ASCII("filename.txt")

rms = pyrana.getRMS()
print("rms = {0:.2f}".format(rms))
```

4.12 loadfile_ASCII

```
PyRANA.loadfile_ASCII(fname)
```

The function `loadfile_ASCII` reads the content of a datafile and stores the wavenumbers and the spectrum in `PyRANA.X` and `PyRANA.Y` arrays. `fname` is the file relative or absolute path. The data file can contain a header section, as long as every lines of the header start with the symbol `'#'`. These lines are simply skipped by the function. Then, the data values are separated by tabulations and ordered in columns corresponding to `X`, `Y`, and optionally `Yerr`.

Example:

```
pyrana = PyRANA()

pyrana.loadfile_ASCII("filename.txt")
```

4.13 loadfile_METAFILE

```
PyRANA.loadfile_METAFILE(fname)
```

The function `loadfile_METAFILE` reads the content of a SALSA metafile, used to import data into SALSA, and stores the wavenumbers and the spectrum in `PyRANA.X` and `PyRANA.Y` arrays. `fname` is the file relative or absolute path. More details on the METAFILE file format can be found on XXXX.

Example:

```
pyrana = PyRANA()

pyrana.loadfile_METAFILE("filename.meta")
```

4.14 loadfile_SALSA

```
PyRANA.loadfile_SALSA(fname)
```

The function `loadfile_SALSA` reads the content of a file exported from SALSA and stores the wavenumbers and the spectrum in `PyRANA.X` and `PyRANA.Y` arrays. `fname` is the file relative or absolute path. More details on the SALSA export file format can be found on XXXX.

Example:

```
pyrana = PyRANA()

pyrana.loadfile_SALSA("salsa-export.csv")
```

4.15 log

```
PyRANA.log(output="pyranaLog.txt")
```

The function log generate an ASCII log file resuming all the fits of all the spectra that have been fitted since the assignation of the `PyRANA` class-object. The file is tabulation-separated and includes a header section with each line beginning by "#". The output file name can be personalized by the `output` parameter.

Example:

```
pyrana = PyRANA()
#
# ...  fit sessions
#
pyrana.log()
```

4.16 modelWrapper

```
PyRANA.modelWrapper(X, modeldef, bound_x0=20, bound_s=20, bound_amin=0,
bound_amax=1, comment=False)
```

The function `modelWrapper` builds the model function depending on the model definition contained in `modeldef`. The `modeldef` parameter is a list of function definition arrays starting with the name of the function and the parameters. `modeldef` can have up to 6 function definitions, the first one defines the continuum function '`poly1`' or '`poly2`', the other functions can be '`cauchy`', '`gauss`', '`assymCauchy`', '`assymGauss`', and '`BWFGraphene`'.

The function returns the built model, an array concatenated of all the parameters in the function definitions, and a `bounds` array, all three formatted for `scipy.curve_fit`. The bound for:

- polynomial parameters: [`-numpy.inf, numpy.inf`],
- intensity: [`0, numpy.inf`],

- position: [x0 - bound_x0, x0 + bound_x0],
- width: [s - bound_s, s + bound_s],
- asymmetry: [bound_amin, bound_amax],

If comment is True, additional information during the execution of the function are printed-out.

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

fct, p0, bounds = pyrana.modelWrapper(X, [ ['poly2', 10, 1, 0.1], ['gauss',
13, 0.2, 2.1] ])
Y = fct(X, *p0)
```

4.17 plot_all

```
PyRANA.plot_all(fits=None, output=None, dpi=96)
```

The function plot_all create a figure showing the entire spectrum. The name of the output figure can be put in the output parameter. A group of fit-type objects can be overlaid on the spectrum by filling the list fits.

Example:

```
pyrana = PyRANA()
pyrana.loadfile_SALSA("salsa-export.csv")
fit = pyrana.quickfit(Xmin=100, Xmax=300, peaks=[
'profile':  'cauchy', 'x0':  150,
'profile':  'cauchy', 'x0':  220 ])

# default plotting
pyrana.plot_all()

# overlay plotting
pyrana.plot_all(fits=[fit])
```

4.18 plot_fit

```
PyRANA.plot_fit(fit, output=None, dpi=96, ascii_output=None)
```

The function plot_fit make a figure showing the fitted region of the spectrum described by the fit-type object fit. The figure can be saved as a PNG or PDF image if the output image name is filled in the output parameter. Alternatively, an ASCII version of the plot can be save by filling the name of the output file name in the ascii_output parameter.

Example:

```
pyrana = PyRANA()
pyrana.loadfile_SALSA("salsa-export.csv")
fit = pyrana.quickfit(Xmin=100, Xmax=300, peaks=[
'profile':  'cauchy', 'x0':  150,
'profile':  'cauchy', 'x0':  220 ])

# default plotting
pyrana.plot_fit(fit)

# saved image
pyrana.plot_fit(fit, output="plot.pdf")

# ascii output
pyrana.plot_fit(fit, ascii_output="plot.txt")
```

4.19 plot_peaks

```
plot_peaks(self, X, Y, bestfit, output=None, dpi=96, ascii_output=None)
```

The function `plot_peaks` make a figure showing the fitted region of the spectrum `X` and `Y`, and `bestfit`, a modelParam structure. The figure can be saved as a PNG or PDF image if the output image name is filled in the `output` parameter. Alternatively, an ASCII version of the plot can be save by filling the name of the output file name in the `ascii_output` parameter.

Example:

```
pyrana = PyRANA()
pyrana.loadfile_SALSA("salsa-export.csv")
fit = pyrana.quickfit(Xmin=100, Xmax=300, peaks=[
'profile':  'cauchy', 'x0':  150,
'profile':  'cauchy', 'x0':  220 ])

# default plotting
pyrana.plot_peaks(fit['dataX'], fit['dataY'], fit['bestfit'])

# saved image
pyrana.plot_peaks(fit['dataX'], fit['dataY'], fit['bestfit'],
output="plot.pdf")

# ascii output
pyrana.plot_peaks(fit['dataX'], fit['dataY'], fit['bestfit'],
ascii_output="plot.txt")
```

4.20 poly1

```
PyRANA.poly1(X, a, b)
```

The function `poly1` returns the linear function evaluation of `X`, a single value or a `numpy.array`:

$$\text{poly1}(X) = a + bX \tag{10}$$

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

Y = pyrana.poly1(X, 1, 0.1)
```

4.21 poly2

```
PyRANA.poly2(X, a, b, c)
```

The function `poly2` returns the polynomial function evaluation of `X`, a single value or a `numpy.array`:

$$\text{poly1}(X) = a + bX + cX^2 \tag{11}$$

Example:

```
X = numpy.linspace(-10, 10, num=1000)
pyrana = PyRANA()

Y = pyrana.poly2(X, 1, 0.1, 0.2)
```

4.22 print_peaks

```
PyRANA.print_peaks(bestfit, besterr)
```

The function `print_peaks` prints out nicely the result of a fit. The best fit parameters and the uncertainties are given by the bestfit and `besterr` modelParam-type variables, respectively. `bestfit` and `besterr` can be found in the the `fit` dictionary returned by the `quickfit` function.

Example:

```
pyrana.PyRANA()
pyrana.loadfile_SALSA("salsa-export.csv")
fit = pyrana.quickfit(Xmin=100, Xmax=300, peaks=[
{'profile':  'cauchy', 'x0':  150},
{'profile':  'cauchy', 'x0':  220} ])

pyrana.print_peaks(fit['bestfit'], fit['besterr'])
```

4.23 quickfit

```
PyRANA.quickfit(X=None, Y=None, errY_=None, continuum = 'linear', peaks =
[{'profile':  'cauchy', 'x0':  None, 'I': None, 's':  None}], Xmin = None,
Xmax = None, use_diffXcontrib=True)
```

The function `quickfit` is the main function of the library. It realize the fit of the loaded spectrum over the spectral window [`Xmin`, `Xmax`] based on the `continuum` and the model `peaks` definition. The function returns the results of the fit as a `fit` structure dictionary.

By default, if nothing is passed to the `X`, `Y`, or `errY`, the function uses the data that has been loaded before. You can replace the three arrays, respectively, as long as they have the same size, otherwise, the program will fail. The `continuum` parameter corresponds to the local continuum

background to use in the fit. The available continuum background function are: 'linear' and 'poly2', for a linear continuum and a second order polynomial continuum, respectively.

The peaks list defines the number of peaks and their shape to fit in the spectral window. peaks can includes up to five peaks, each of them defined with a dictionary, as follow:

```
peaks = [
{'profile':  str, # MANDATORY, available profile:  'cauchy', 'gauss',
'assymCauchy', 'assymGauss', and 'BWFGraphene'
'x0':  float, # optional but recommended
'I': float, # optional
's':  float, # optional
'a':  float, # optional, only used for asymmetric profiles
},
...
]
```

It is recommended to always put the name of the profile and the position in the peaks definition. If you have a good idea of the other parameters, you can put them as well, but it is not necessary. The parameters filled in the peak definition are sent to scipy.curve_fit as a first guess for the fit. The default values, when nothing is filled for the parameters, are: 'x0': (Xmax – Xmin) / 2, 'I': 1000, 's': 20, and 'a': 0.1.

The fit algorithm searches the best set of peak parameters that fits the spectrum over constrained ranges for the position, the width and the asymmetric factor, whereas the intensity is just set to be positive. The searching ranges are:
- position: [x0–20, x0+20],
- width: [s–20, s+20],
- asymmetric factor: [0, 1].

The last parameter use_diffXcontrib enable the fit to use the X-axis binning into account in the uncertainties. This contribution is usually negligible for well defined spectrum. However, the X-axis binning becomes the major uncertainty contribution when the peak spawn over only a few bins. The fit is made two times: the first time taking only the Y-axis uncertainties σ_{y} into account, the second time using the first best-fit model f_{fit} and the uncertainties along X- and Y-axes, σ_{x} and σ_{y}, as follow:

$$\sigma_{\mathrm{tot}}^2 = \sigma_{\mathrm{y}}^2 + \left(\frac{\mathrm{d}f_{\mathrm{fit}}}{\mathrm{dx}}\right)^2 \sigma_{\mathrm{x}}^2 \tag{12}$$

When considering the X-axis binning contribution in the uncertainty, the fit is usually not significantly affected by the contribution, however, the error on the best fit parameters is increased and reflects properly the poor spectral resolution of the data. Figure 4 shows an example of binning contribution for the uncertainty of a weak binning Gaussian peak.

Example:

```
pyrana = PyRANA()
pyrana.loadfile_ASCII('filename.txt')

# Default quickfit use
pyrana.quickfit(Xmin=100, Xmax=300, peaks=[
{'profile':'cauchy', 'x0':150},
{'profile':'cauchy', 'x0':220} ])
```

```
 # quickfit alternative example
pyrana.quickfit(Xmin=200, Xmax=380, continuum='poly2', peaks=[
{'profile':'cauchy', 'x0':150, 's':40},
{'profile':'assymCauchy', 'x0':220} ])
```

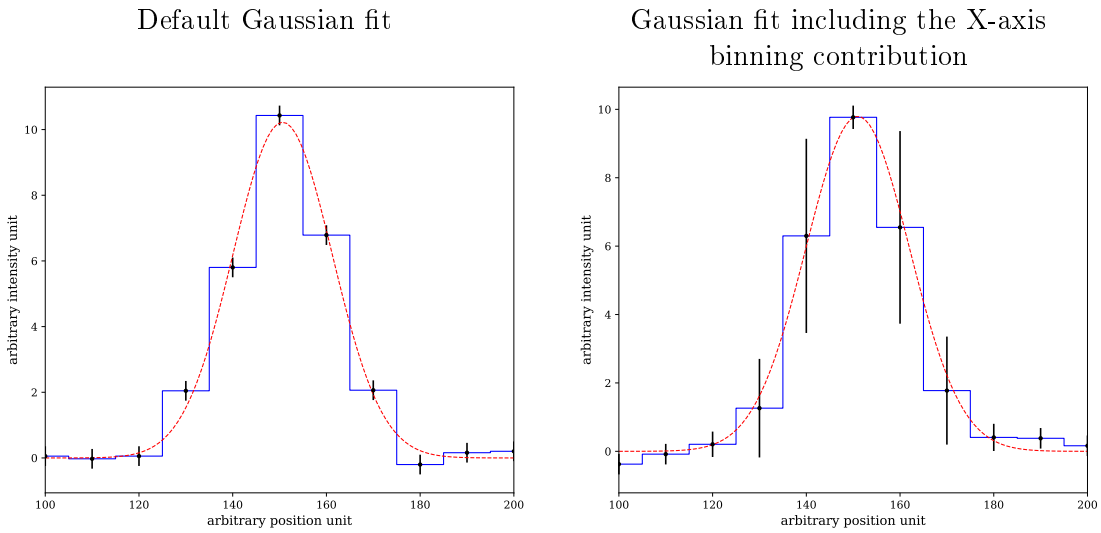

Figure 4: Example of Gaussian fit without and with the X-axis binning contribution taken into account, on the left and right panel, respectively. The blue and red dashed curves correspond to the data and the fit model, respectively. The uncertainties are shown as black straight lines in the middle of each bin.

4.24 str_peaks

```
PyRANA.str_peaks(fits, typetxt='ascii')
```

The function `str_peaks` formates the fit results of each fit of the list `fits`. `typetxt` is a word defining the format of the output str. The available format are:

- 'ascii': this produces a tabulation-separated table of the fitted parameters, errors, chi2, number of points, and the probability of the chi-square survival function. The format corresponds to the output of the `log` function.

Example:

```
pyrana.PyRANA()
pyrana.loadfile_SALSA("salsa-export.csv")
fit = pyrana.quickfit(Xmin=100, Xmax=300, peaks=[
{'profile':  'cauchy', 'x0':  150},
{'profile':  'cauchy', 'x0':  220} ])

pyrana.str_peaks([fit])
```

5 Bibliography

References

P. C. Eklund and K. R. Subbaswamy. Analysis of Breit-Wigner line shapes in the Raman spectra of graphite intercalation compounds. *Phys. Rev. B*, 20(12):5157–5161, December 1979. doi: 10.1103/PhysRevB.20.5157.

Eddwi H. Hasdeo, Ahmad R. T. Nugraha, Mildred S. Dresselhaus, and Riichiro Saito. Breit-wigner-fano line shapes in raman spectra of graphene. *Phys. Rev. B*, 90:245140, Dec 2014. doi: 10.1103/PhysRevB.90.245140. URL `https://link.aps.org/doi/10.1103/PhysRevB.90.245140`.

John D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science and Engineering*, 9(3):90–95, May 2007. doi: 10.1109/MCSE.2007.55.

E. Jones, T. Oliphant, and P. Peterson. Scipy: Open source scientific tools for python. *online*, 2001.

M. V. Klein. *Light scattering in Solids 1*, volume 8. Springer-Verlag Berlin Heidelberg, 1975. ISBN 978-3-540-37568-5. doi: 10.1007/978-3-540-37568-5.

Aaron L. Stancik and Eric B. Brauns. A simple asymmetric lineshape for fitting infrared absorption spectra. *Vibrational Spectroscopy*, 47(1):66–69, 2008. ISSN 0924-2031. doi: https://doi.org/10.1016/j.vibspec.2008.02.009. URL `https://www.sciencedirect.com/science/article/pii/S0924203108000453`.

Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science and Engineering*, 13(2):22–30, March 2011. doi: 10.1109/MCSE.2011.37.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, February 2020. doi: 10.1038/s41592-019-0686-2.

6 Appendix: utility objects

6.1 modelParam

The `modelParam` structure array describes the profiles of the peaks included in the model and the parameters of the best fit. The type-def of `modelParam` is shown below:

```
modelParam = [
[str1, par1_1, par2_1, par3_1], # peak definition with 2 to 4 parameters
[str2, par1_2, par2_2, par3_2, par4_2], # peak def.  with 2 to 4 parameters
[str3, par1_3, par2_3, par3_3] ] # peak definition with 2 to 4 parameters
```

6.2 fit

The `fit` structure dictionary describes the result of a fit made by the function `quickfit`. It includes the data used for the fit, the uncertainties, the modelParam of the best fit, the errors on the best fit parameters in the format of a `modelParam` structure, and the modelled spectrum. The type-def of `fit` is shown below:

```
fit = {
'bestfit':  [
[str1, par1_1, par2_1, par3_1],
[str2, par1_2, par2_2, par3_2, par4_2],
[str3, par1_3, par2_3, par3_3] ],
'errors':  [
[str1, err1_1, err2_1, err3_1],
[str2, err1_2, err2_2, err3_2, err4_2],
[str3, err1_3, err2_3, err3_3] ],
'dataX': X, # numpy.array
'dataY': Y, # numpy.array
'dataYerr':  uncY, # numpy.array
'model':  fitY } # numpy.array
```