

NRCA-Spectra

Last revision: October 2020.

This is a program written by Ivan Alsina Ferrer (ivanalsinaferrer@gmail.com) during a summer internship at STFC (Oxfordshire, UK) on the period Jun-Aug 2019.

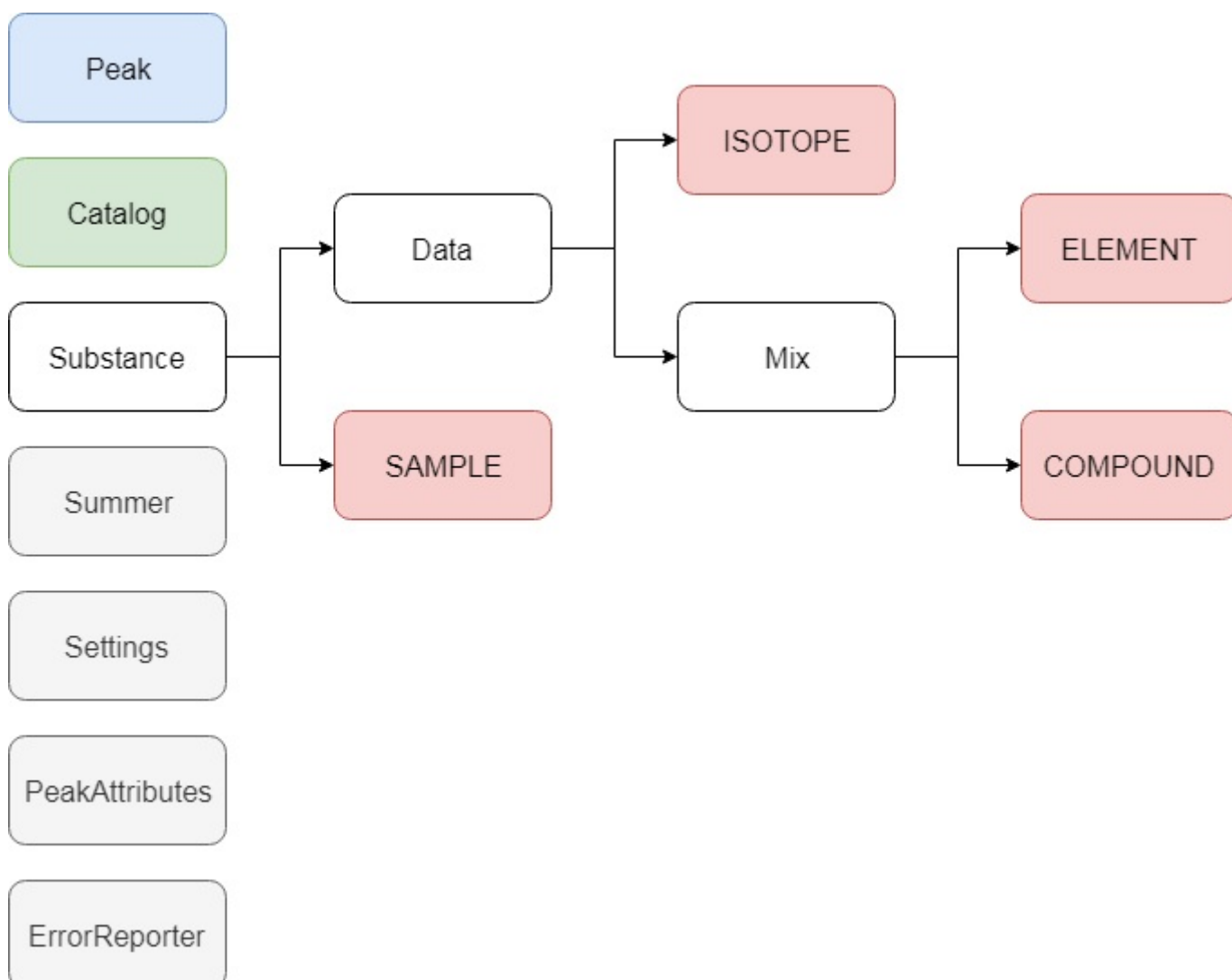
It intends to create a database of spectra that stores information on each needed **Isotope**, is able to weight them to create **Elements** and **Compounds**, and stores **Sample** spectra from the INES instrument at STFC. It also includes some functions that help import data, weigh it, process it, and correlate it with the experimental sample outcome.

Code overview

Inheritance strategy

In the previous paragraph, words in bold are particularly important terminology-wise because they are the main elemental objects that carry information on items to be studied.

There are other parent classes conceived for code reusability's sake, namely Substance, Data and Mix. This is because some of them share important methods and attributes (M&A). There are as well other classes used for the information processing that don't share these M&A. In terms, merely, of this feature hierarchy, all the class objects are built as follows:



All the objects with the same indentation are at the same level, which means that independently defined. However, objects indented inherit M&A. For this reason, every M&A defined for Data are common for **Isotope**, **Element** and **Compound**, but aren't seen by **Sample**. Similarly, every M&A defined for Mix can be called by **Element** and **Compound**, but not by **Isotope**.

Another important class object is **Peak**. This stores relevant information for a particular peak of an **Isotope**, an **Element** or a **Compound**, so only Data objects. Their M&A are alone defined for itself.

Another very relevant class object is **Catalog**. This is the big container that is created (or loaded) at the beginning of the program and stores every bit of relevant information. It also has M&A that gives access to the program information and functionalities. The program is designed so that there is need to import an instance of this **Catalog**. Everything is within it.

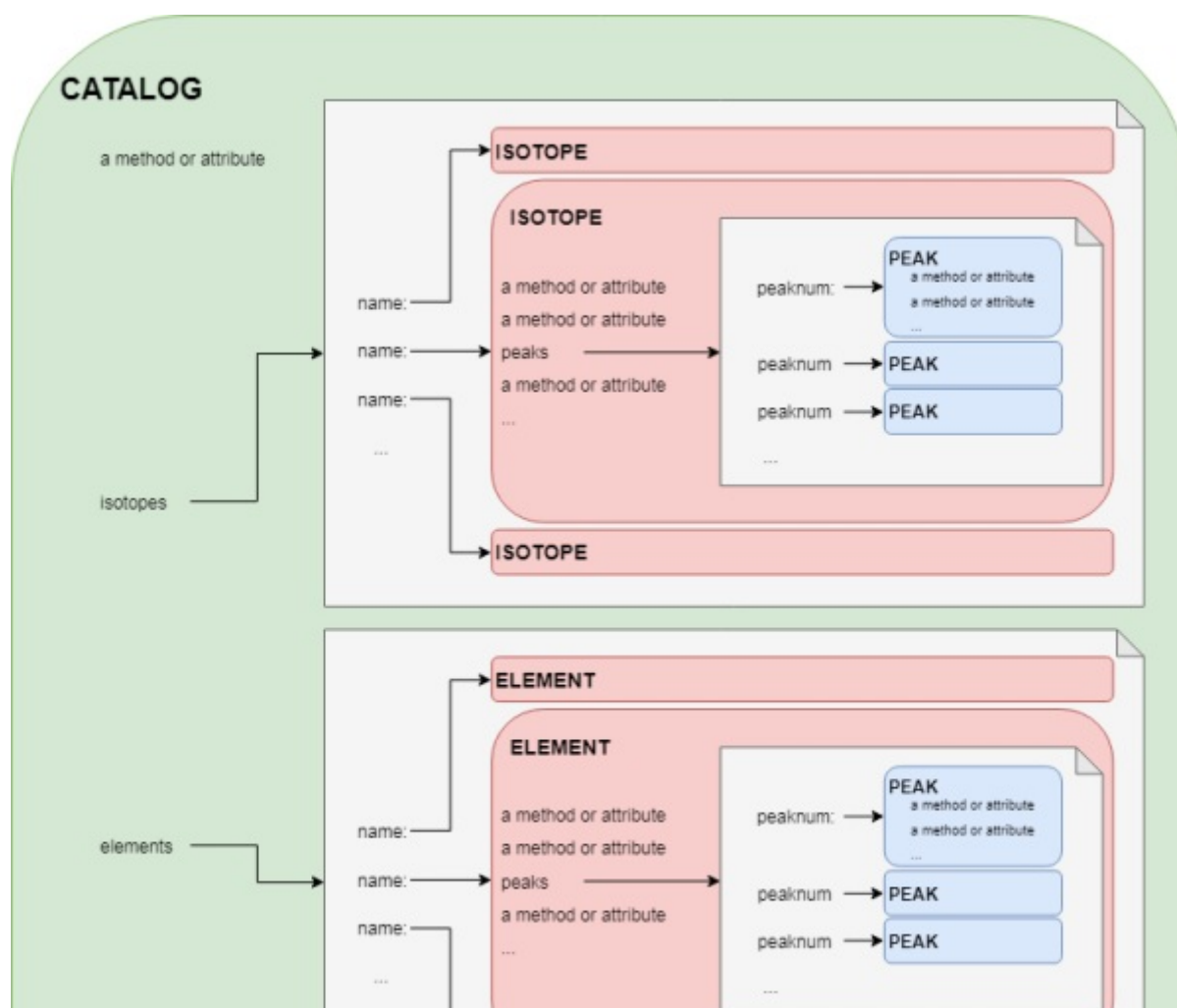
Throughout the documentation, the word **instance** means 'python variable with entity that is built after a class and stores certain data' (see §§ Instantiation). The feature hierarchy is as follows (capitals indicate actual usable classes.) Thus, 'a Data instance' means 'either an Isotope, an Element or a Compound instance. The other classes are a little less relevant for the user and we will be skipping them for now.

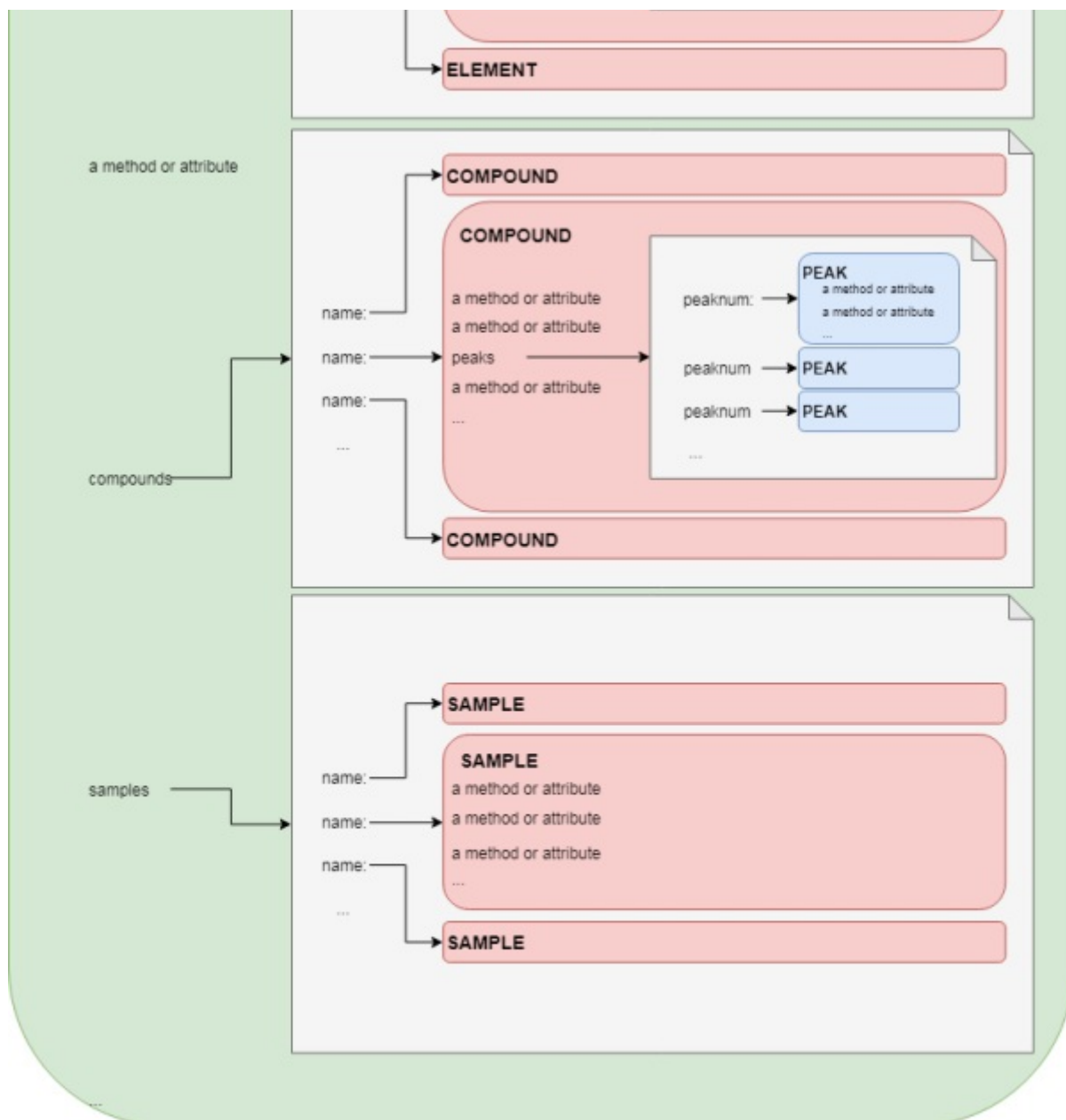
Storage strategy

Then, how is the information stored? How can it be accessed and modified?

Every **Substance** instance is stored in a python dictionary, as an attribute of its *container class*. Now we are facing a new kind of hierarchy that has nothing to do with the previous one. This hierarchy (probably more relevant and understandable) has to do with the structure scheme of information of the database. It tells us what contains what.

In terms, now, of structure hierarchy, the class objects are organised as follows:





Thus, the **Catalog** contains a dictionary of **Isotopes**, another dictionary of **Elements**, another of **Compounds** and a last one of **Samples**. Each **Isotope** contains a dictionary of **Peaks**, and the same applies to **Elements** and **Compounds**.

Creation and detection strategy

Whenever an isotope, element or compound file is read from the data/ (or equivalent -- see §§ Setup) a **Data** object is created. Then, the following sequence follows:

1. The energy spectrum is read and stored as energy spectrum (`. spectrum`) and as ToF spectrum (`. spectrum_tof`).
2. The spectrum maxima values are computed. Amongst them, only those whose cross-section is above the minimum (**crs_min**) and whose energy is between the thresholds (**thr_min**, **thr_max**) are stored as an array in an attribute (`. ma`). The same applies to their indices (`. mai`) and the maxima of the ToF spectrum (`. ma_tof`). Then, the number of peaks that there are is stored (`. npeaks`).
3. The derivative (`. der`) is computed and stored. Then, in all the outermost left points where its absolute value is greater than **maxleftslope**, it is patched and set to 0.
4. A smoothed derivative (`. sder`) is computed and stored. An integer **itersmooth** controls how much it smoothes.
5. Peaks are built and stored.

- If a dictionary of peaks is passed, i.e. we are importing peaks from a load/peakprops .txt file, it is stored.
- Otherwise, peaks are computed and **Peak detection** begins. In that case, for every peak:
 1. A window of size $2 \times \text{prange} + 1$ (named *redder - reduced derivative*) centered to the peak maximum is considered. Here, *prange* is taken as the minimum between the distance to the next peak, the number of spectrum mesh points at the left of the peak, the number of spectrum mesh points at the right of the peak, and **prangemax**.
 2. A histogram of slope (frequency) occurrence, with bar (or box) density (**dbboxes**) is computed, and the most frequent slope is named *outerslope*. This is considered to be the slope away from the peak.
 3. The pointer is set on the maximum.
 4. Moving the pointer to the left, and keeping track of the (smoothed) derivative, whenever the derivative starts decreasing (the peak could still be flat at the top), a **lock** flag is raised, at that point we are at the maximum derivative, *dermax*. If, when this happens, *dermax* equals *outerslope*, an error is raised (*non-standing slope*).
 5. After the **lock** flag is raised, whenever one of the following conditions is fulfilled, the peak is bounded.
 - **Cond. 1.** The slope has dropped down to a **slopedrop** fraction between *dermax* and *outerslope*. (A value of 1 would mean cut at *dermax*, and value of 0 would mean cut at *outerslope*).
 - **Cond. 2.** Next derivative sign times the current derivative sign is lower or equal than 0, i.e. the peak flattens, or a minimum is found.
 6. Repeat steps 3-5, but now moving to the right and switching decreasing for increasing.
 7. If at some point in the process, the pointer reaches the end of the window, an error is raised (*unable to define peak*). Similarly, if the peak boundaries have the same value, an error is raised (*zero-width peak*).

- **Remarks about the computation**

- The variables presented in **bold typefont** can be set in the settings file.

- **Remarks about peaks**

- The numerical properties (atributes) computed for the peaks are: *.center* (eV), *.integral* (intensity, b*eV), *.width* (eV), *.height* (b), *.fwhm* (eV), *.ahh* (integral/height²), *.ahw* (integral/(height*width)).
 - If the peaks are computed, they are sorted by integral (higher to lower). Then, they are labeled accordingly, and stored. These labels start with 0. This means that in the peaks dictionary of, say, an isotope, the 0th element of the peaks dictionary will be the most intense one; the 1st, the next one in intensity, and so on.
 - If the user ever changes the peak definition safely (see below), the peaks are again sorted and renamed accordingly. A flag is stored for an edited peak if this happens.

6. The unsuccessful peaks are stored (*.errors* attribute).

User's manual

Instantiation and pointing

In python, a variable can point at a certain initialised class. In that sense, an instance for each Isotope, Element, Compound and Sample, is stored in the respective dictionaries of contained in the Catalog class. Besides that, a new variable can be pointed at any of the existing instances.

Setup

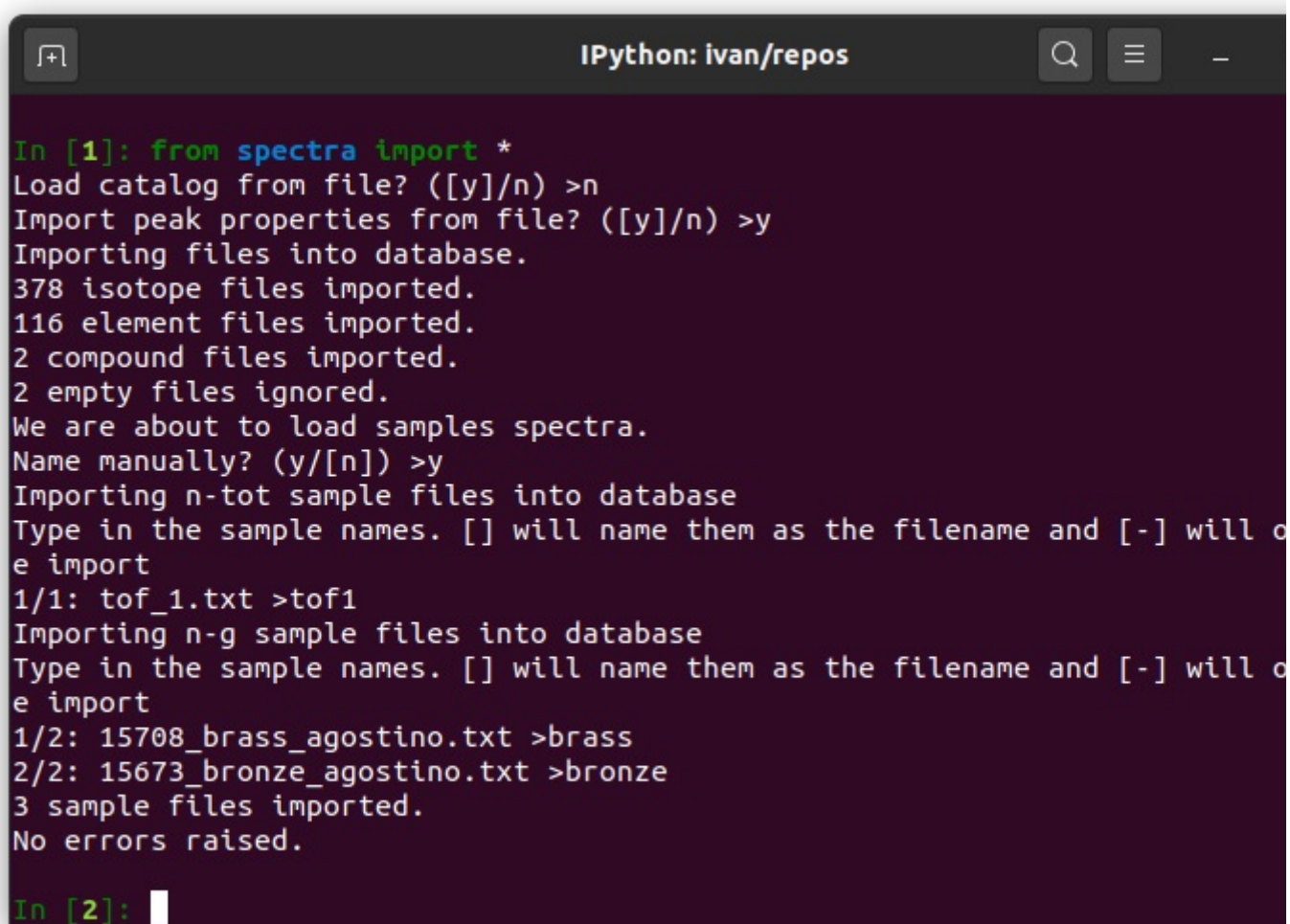
To begin to use the project, the user needs to:

1. Clone the repository from github.
2. Rename the directory to `spectra` and place it wherever.
3. The paths .txt file contains the paths to the relevant work directories. By default they are stored within the same directory, but it can be changed to any route in the system. The entry `cwd` shouldn't be changed, though.

Initialization

1. Initialize a Python console (or run a script) in the directory above `spectra`, i.e., where `spectra` is contained.
2. Run the line

```
from spectra import *
```



```
IPython: ivan/repos

In [1]: from spectra import *
Load catalog from file? ([y]/n) >n
Import peak properties from file? ([y]/n) >y
Importing files into database.
378 isotope files imported.
116 element files imported.
2 compound files imported.
2 empty files ignored.
We are about to load samples spectra.
Name manually? (y/[n]) >y
Importing n-tot sample files into database
Type in the sample names. [] will name them as the filename and [-] will o
e import
1/1: tof_1.txt >tof1
Importing n-g sample files into database
Type in the sample names. [] will name them as the filename and [-] will o
e import
1/2: 15708_brass_agostino.txt >brass
2/2: 15673_bronze_agostino.txt >bronze
3 sample files imported.
No errors raised.

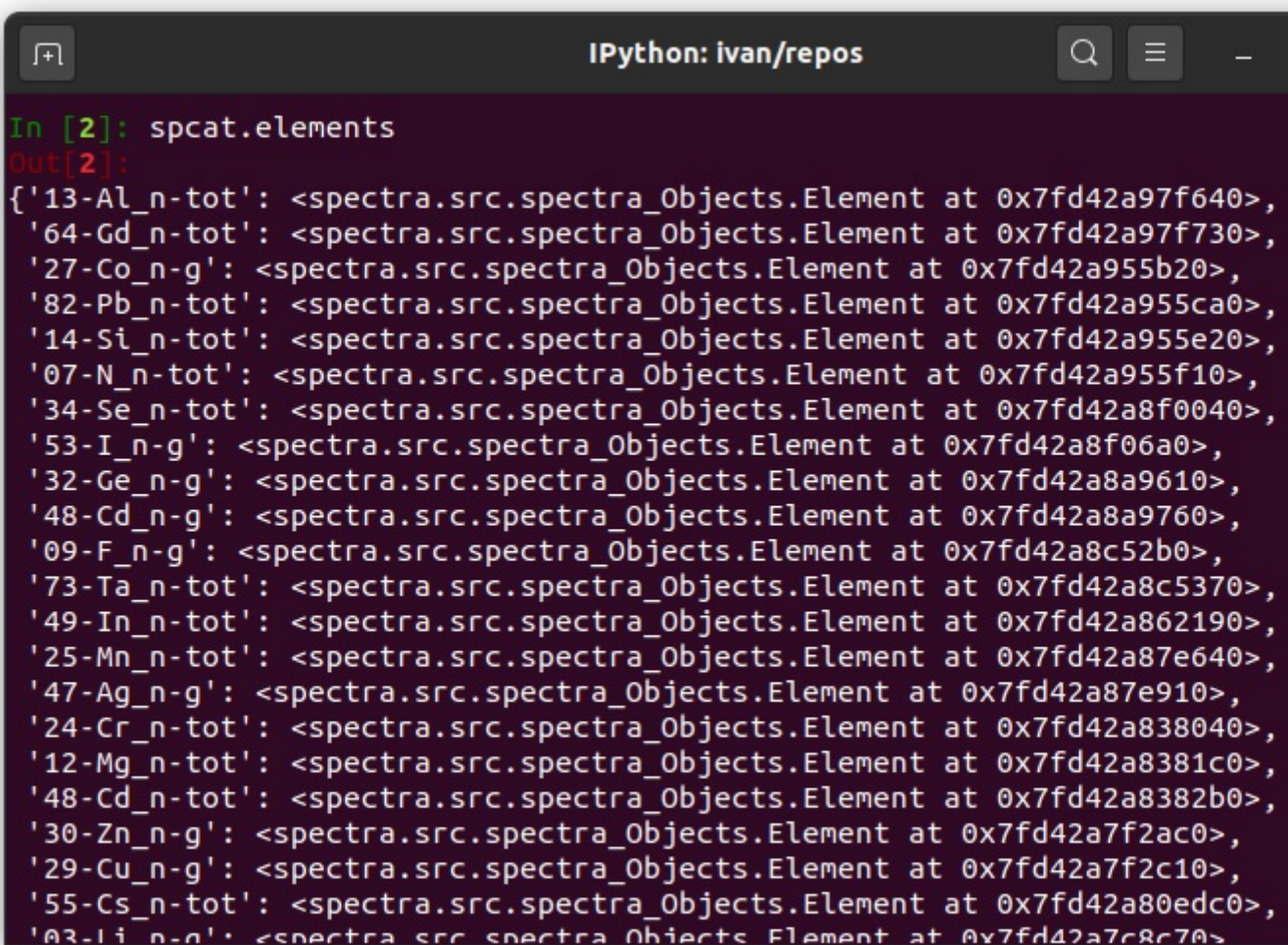
In [2]:
```

In this case, since a `spectra.pickle` object exists (see §§ Saving), the program asks whether it should be loaded or not. If not, we are guided through the importing process.


```
In [1]: from spectra import *
Load catalog from file? ([y]/n) >y
Catalog imported.
Date of creation: 10/10/2020 18:01:54
Date of last modification: 10/10/2020 18:01:58
```

If it exists, then the loading is easier and faster.

Once we have done this, we can access the spcat instance and everything in it. The Catalog instance is named as spcat (*spectra catalogue*).



```
IPython: ivan/repos

In [2]: spcat.elements
Out[2]:
{'13-Al_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a97f640>,
 '64-Gd_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a97f730>,
 '27-Co_n-g': <spectra.src.spectra_Objects.Element at 0x7fd42a955b20>,
 '82-Pb_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a955ca0>,
 '14-Si_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a955e20>,
 '07-N_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a955f10>,
 '34-Se_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a8f0040>,
 '53-I_n-g': <spectra.src.spectra_Objects.Element at 0x7fd42a8f06a0>,
 '32-Ge_n-g': <spectra.src.spectra_Objects.Element at 0x7fd42a8a9610>,
 '48-Cd_n-g': <spectra.src.spectra_Objects.Element at 0x7fd42a8a9760>,
 '09-F_n-g': <spectra.src.spectra_Objects.Element at 0x7fd42a8c52b0>,
 '73-Ta_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a8c5370>,
 '49-In_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a862190>,
 '25-Mn_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a87e640>,
 '47-Ag_n-g': <spectra.src.spectra_Objects.Element at 0x7fd42a87e910>,
 '24-Cr_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a838040>,
 '12-Mg_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a8381c0>,
 '48-Cd_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a8382b0>,
 '30-Zn_n-g': <spectra.src.spectra_Objects.Element at 0x7fd42a7f2ac0>,
 '29-Cu_n-g': <spectra.src.spectra_Objects.Element at 0x7fd42a7f2c10>,
 '55-Cs_n-tot': <spectra.src.spectra_Objects.Element at 0x7fd42a80edc0>,
 '03-Li_n-g': <spectra.src.spectra_Objects.Element at 0x7fd42a7c8c70>}
```

Accessing and pointing

More interestingly, to reach a particular substance, we can use the method `get()`.

```
IPython: ivan/repos

In [3]: iridium = spcat.get('77-Ir_n-g')

In [4]: iridium
Out[4]: <spectra.src.spectra_Objects.Element at 0x7f4a92fde3d0>

In [5]: iridium.fullname
Out[5]: '77-Ir_n-g'

In [6]: iridium.kind
Out[6]: 'element'

In [7]: iridium.peaks[0]
Out[7]: <spectra.src.spectra_Objects.Peak at 0x7f4a92fde490>

In [8]: iridium.peaks[0].coords
Out[8]: (43.18, 4992.30235)

In [9]: iridium_peak = iridium.peaks[0]

In [10]: iridium_peak
Out[10]: <spectra.src.spectra_Objects.Peak at 0x7f4a92fde490>

In [11]:
```

Here, we use the variable `iridium` to point a particular instance (in this case, an instance of `Element`) of `spcat`, the `Catalogue` instance. Thus, it is possible to see where it is in memory and access any of its attributes. In particular, we can access its peaks (specifically the most intense), and then access *the peak's* attributes. It is also possible to use a new variable to point at the same instance (same place in memory, as we can see).

The method `find()` is even easier to use:

```
IPython: ivan/repos

In [11]: iridium191 = spcat.find()
>Ir
1: 77-Ir-191_n-g
2: 77-Ir-191_n-tot
3: 77-Ir-193_n-g
4: 77-Ir-193_n-tot
5: 77-Ir_n-g
6: 77-Ir_n-tot
Select above results of your query ([0] all; [] None) >1

In [12]: iridium191
Out[12]: <spectra.src.spectra_Objects.Isotope at 0x7f032a43b820>

In [13]: iridium191.kind
Out[13]: 'isotope'

In [14]: iridium191.npeaks
Out[14]: 43

In [15]: iridium191.peaks[0].xlims
Out[15]: (5.19769, 5.50892)

In [16]:
```

Here, we seeked by "Ir" all the possible matches (we could have also seeked "77-Ir" or "Ir-191"), and then selected our choice, the isotope Ir-191.

Peaks

Peaks are stored in a dictionary accessed through attribute peaks, as shown above.

Although the peaks attributes can be accessed the same way, it is **not recommended** to edit them through something like

```
iridium.peaks[0].center = 52.5
```

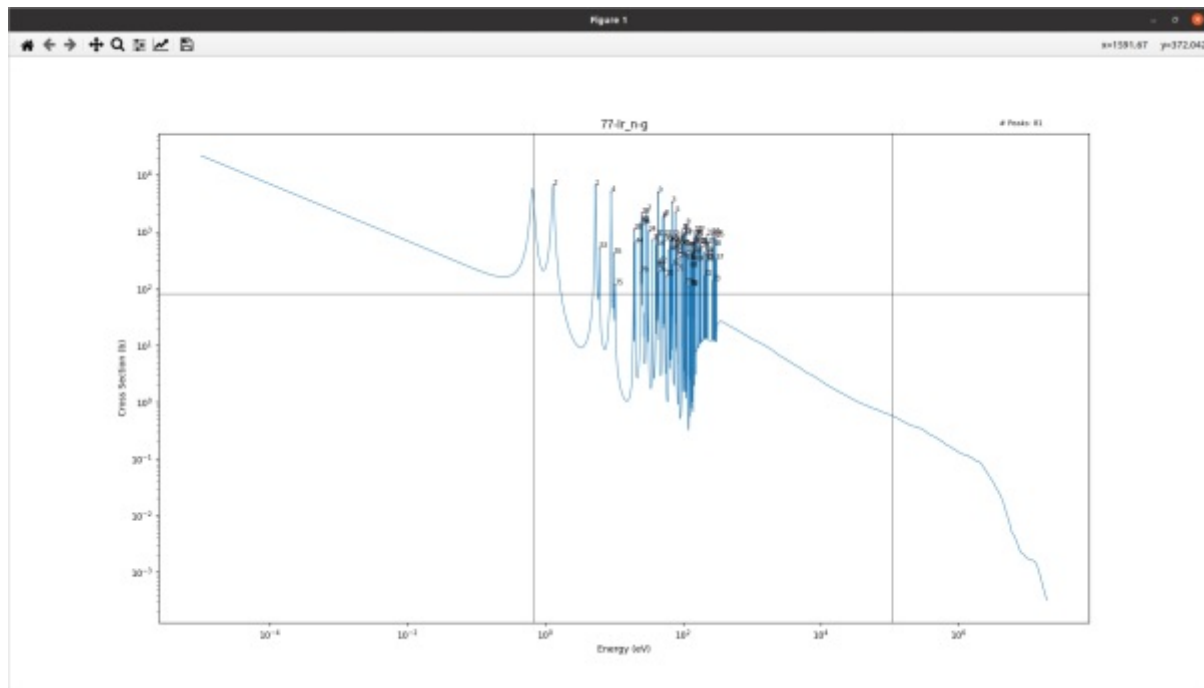
As, in general, editing a peak should involve a series of operations in order to update the other attributes. See §§ Modifying the peak scheme.

The peak attributes are implemented in a class named `PeakAttributes`, used for setting and getting them, and they are shown in § Appendix.

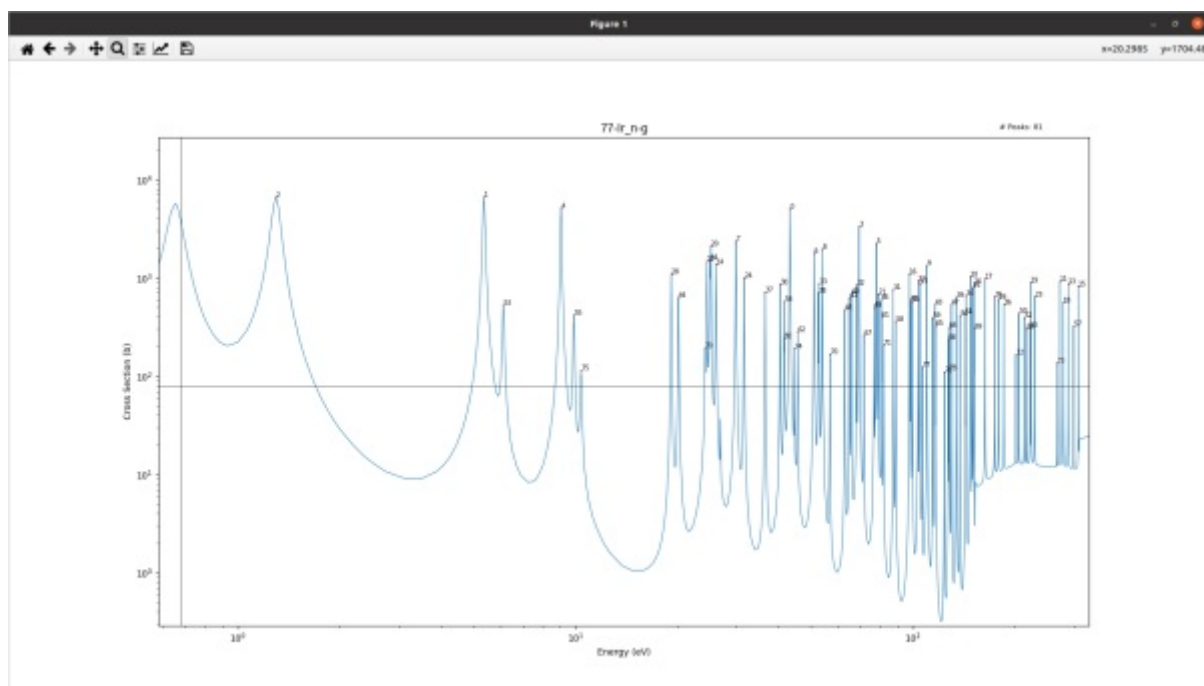
Plotting

Once we have the pointer `iridium` to a Substance, we can plot it by typing:

```
iridium.plot()
```

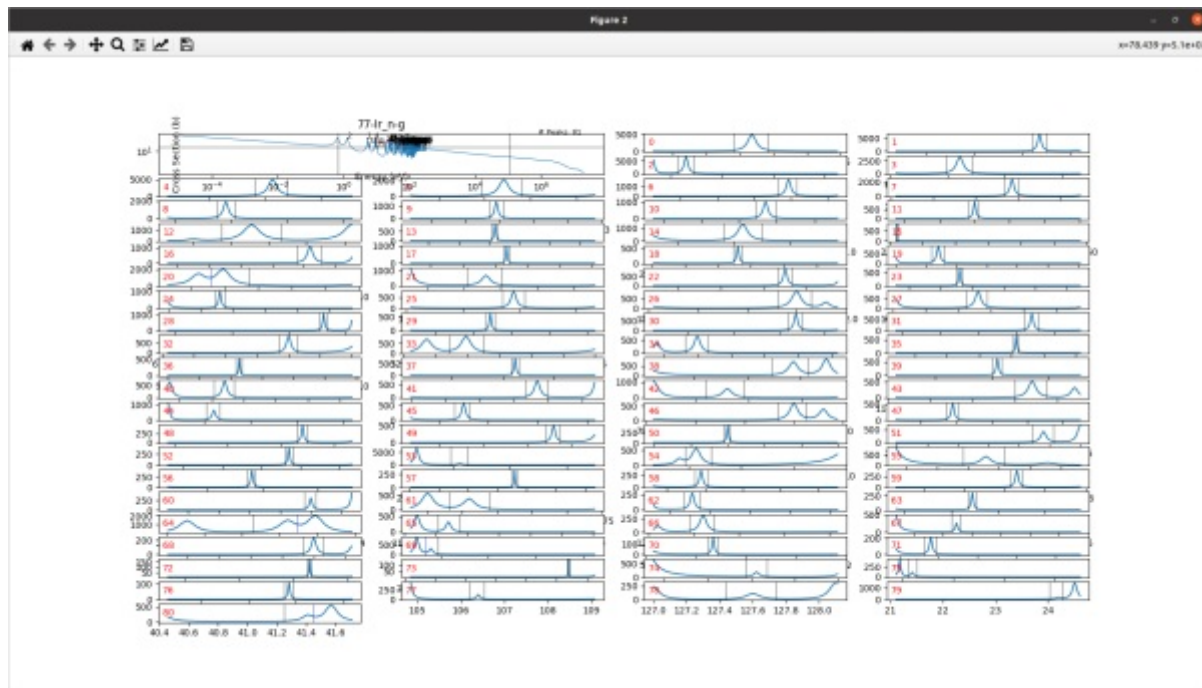
And then use the toolbar of the interactive `matplotlib` window to explore the spectrum. For example, zooming in:



Another interesting option is to have an overview of all the peaks of the sample, in that case

```
iridium.plotpeaks()
```

yields



In this window, all the peaks are shown, along with their center and their borders. It is maybe not the greatest tool to see them properly, but it enables inspection to make sure there are not any wrong detections.

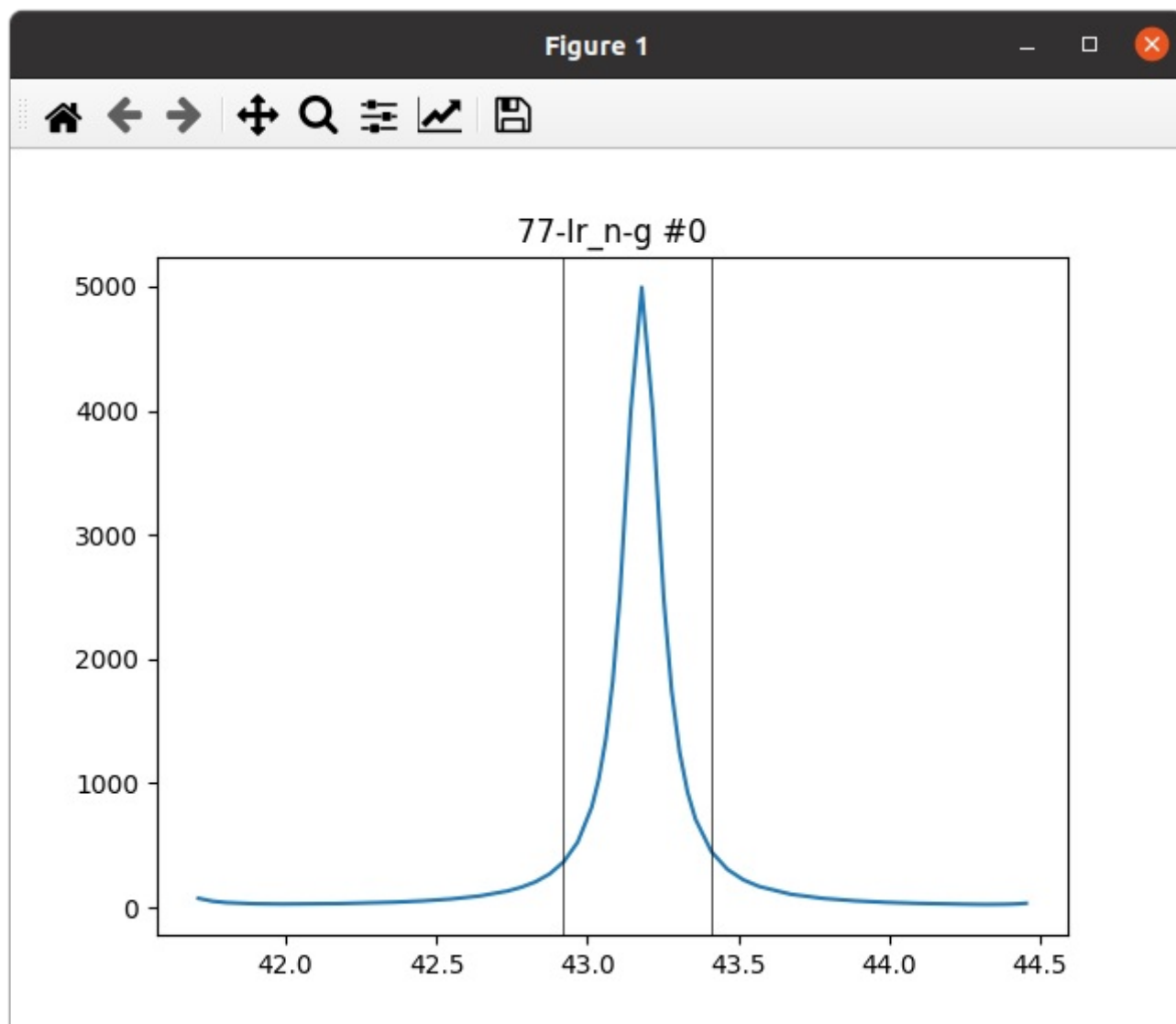
Color code:

- Black lines: normal, bounded through condition 1 (see §§ Detection Strategy).
- Blue lines: normal, bounded through condition 2.
- Green lines: normal, user edited.
- Red line at the center: error, unable to define.

Another possibility is to save the graph by means of the floppy disk at the toolbar.

If we want to plot a particular peak, say the 0th, this is what we should do:

```
iridium.plotsingle(0)
```



Modifying the peak scheme

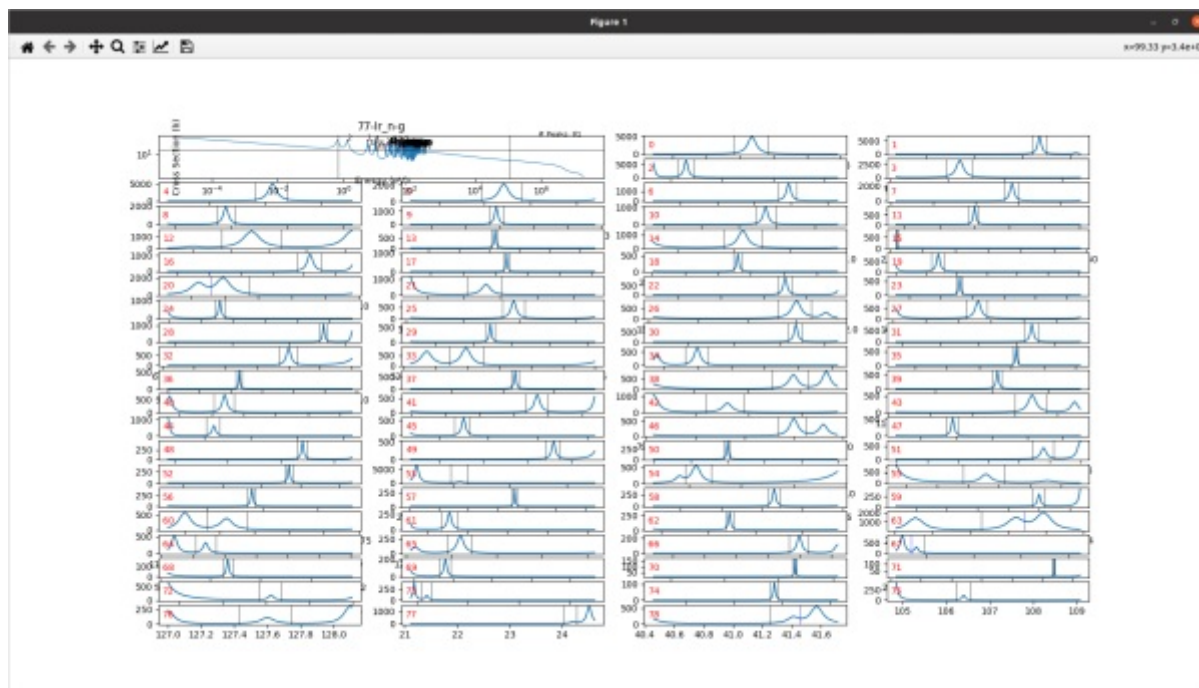
Imagine that we find out that the peaks 58 and 67 have been detected but are false peaks. The way to proceed is the following:

```
iridium.delete()
```

And an interactive process starts:

```
In [3]: iridium.delete()
Enter peak: >58
To delete: [58]
Enter peak: >67
To delete: [58, 67]
Enter peak: >
This action will delete the following peaks:
[58, 67]
Continue? (y/[n]) >y
```

If we now plot the peaks again by means of `iridium.plotpeaks()`, we get:



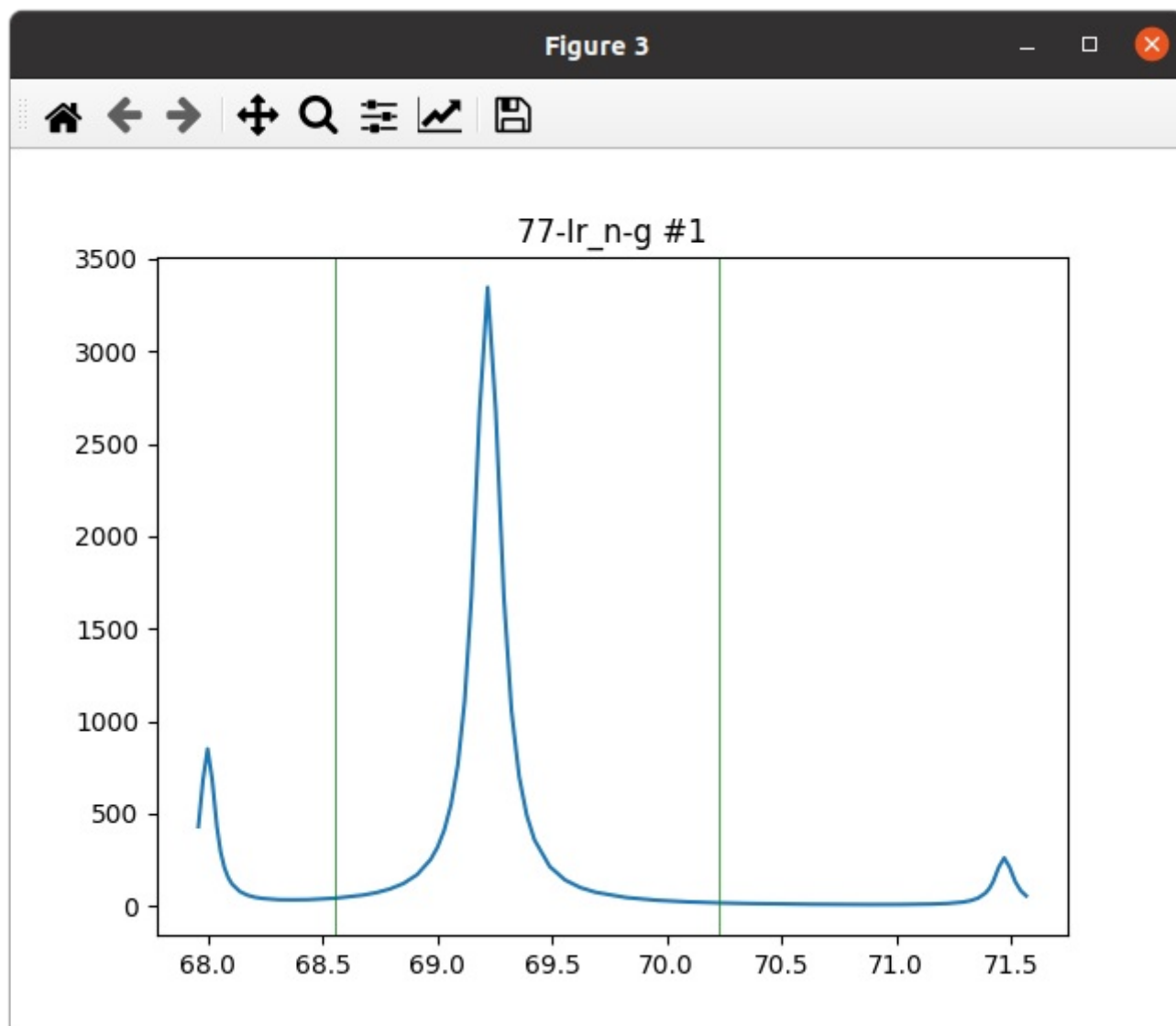
If a peak has been badly defined, the way to edit it is the following:

```
iridium.edit()
```

Then, we go through the following steps:

```
In [6]: iridium.edit()
Enter peak: >3
        Old peak boundaries: (68.9715, 69.4237)
        Enter new peak boundaries: >68.5, 70.2
        New peak boundaries: (68.5563, 70.223)
Enter peak: >
This action will edit the following peaks:
[3]
Continue? ([y]/n) >y
```

Please, note: Every time a peak is deleted or edited, the peak labels change as they are set accordingly to the peak rank in intensity (integral). For instance, we edited the peak #3 (fourth most intense) to make it wider. Now it has passed to be the peak #1 (second most intense). Thus, by calling `iridium.plotsingle(1)`, we see the freshly edited peak:



Viewing information

Anytime, to check the peaks and their basic properties, it is possible to call:

```
iridium.infopeaks()
```

to see:


```
In [11]: iridium.infopeaks()
```

```
77-Ir_n-g PEAKS: =====
Rk.  Energy (eV)      TOF (us)      Integral      Peak width      Peak h
```

0	4.318e+01	(19)	2.514e+02	7.308e+02	4.896e-01 (8)	4.584e+03 (3)
1	6.922e+01	(31)	1.986e+02	6.525e+02	1.667e+00 (0)	3.316e+03 (4)
2	5.360e+00	(1)	7.137e+02	6.044e+02	3.112e-01 (56)	6.187e+03 (6)
3	1.298e+00	(0)	1.450e+03	6.001e+02	3.173e-01 (51)	6.121e+03 (1)
4	9.068e+00	(3)	5.487e+02	4.484e+02	2.874e-01 (67)	4.689e+03 (2)
5	7.771e+01	(34)	1.874e+02	2.803e+02	4.510e-01 (12)	2.080e+03 (6)
6	5.094e+01	(22)	2.315e+02	2.633e+02	5.161e-01 (5)	1.666e+03 (8)
7	2.986e+01	(13)	3.024e+02	2.489e+02	3.443e-01 (33)	2.151e+03 (5)
8	5.391e+01	(25)	2.250e+02	1.995e+02	3.320e-01 (42)	1.768e+03 (7)
9	1.099e+02	(47)	1.576e+02	1.583e+02	4.110e-01 (14)	1.248e+03 (1)
10	1.477e+02	(60)	1.360e+02	1.526e+02	5.140e-01 (6)	9.507e+02 (1)
11	2.720e+02	(76)	1.002e+02	1.450e+02	5.450e-01 (3)	8.425e+02 (2)
12	2.451e+01	(9)	3.337e+02	1.390e+02	3.143e-01 (55)	1.312e+03 (1)
13	2.880e+02	(78)	9.736e+01	1.366e+02	5.330e-01 (4)	7.948e+02 (2)
14	2.608e+01	(12)	3.235e+02	1.357e+02	3.369e-01 (38)	1.250e+03 (1)
15	3.090e+02	(80)	9.400e+01	1.300e+02	5.610e-01 (2)	7.456e+02 (2)
16	9.760e+01	(41)	1.672e+02	1.166e+02	3.737e-01 (23)	1.015e+03 (1)
17	1.634e+02	(64)	1.293e+02	1.141e+02	4.020e-01 (15)	8.970e+02 (1)
18	2.790e+02	(77)	9.892e+01	1.106e+02	6.730e-01 (1)	5.063e+02 (4)
19	2.234e+02	(73)	1.105e+02	1.094e+02	4.140e-01 (13)	8.044e+02 (2)
20	2.519e+01	(11)	3.292e+02	1.053e+02	2.248e-01 (79)	1.497e+03 (9)
21	7.880e+01	(35)	1.861e+02	9.493e+01	4.811e-01 (9)	6.260e+02 (3)
22	1.035e+02	(44)	1.624e+02	9.413e+01	3.480e-01 (31)	8.547e+02 (1)
23	2.290e+02	(74)	1.092e+02	9.236e+01	5.070e-01 (7)	5.757e+02 (3)
24	3.159e+01	(14)	2.940e+02	9.219e+01	3.273e-01 (45)	9.324e+02 (1)
25	1.499e+02	(61)	1.350e+02	9.035e+01	3.950e-01 (18)	7.436e+02 (2)

Namely, a list of the peaks (obviously ranked by intensity - integral, which is their name), as long as their energy (center), ToF, width and height. In parentheses we see how the peaks would rank in each of the properties. Thus, in the example above, the peak #3 is the first in energy, i.e. the outermost left in the energy spectrum.

Saving

To store the information in spcat as it is at a certain point, the following needs to be done:

```
spcat.save()
```

```
spcat.export()
```

The first command dumps spcat in a file named spcat.pickle (in the load directory) and the second one creates the files PeakProperties.txt (in the main directory) and peakprops.txt (in the load directory). PeakProperties.txt is a file with the same nicely-shown information as the one in the picture above, and peakprops.txt contains all the information for every peak. The latter is non human readable but very useful to store properties and load them without having to compute them every time. This is useful when pickle or the spcat.pickle file is not available.

Recompute

If, instead we want to undo the changes we made in a Data instance, we can use the following command:

```
iridium.recompute()
```

Interestingly, if we want to re-compute with a specific set of parameters, we can pass them as arguments. The ones that aren't passed, are picked from the `settings.py` file. For example:

```
iridium.recompute(slopedrop=0.65, prangemax=800)
```

Remarks:

- If an argument `e` is passed, it indicates the xbounds (threshold for peak detection) and *must* be a tuple, e.g.

```
iridium.recompute(e = (20, 2000))
```

- If an argument `tof` is passed, it is converted into energy first (since the Data spectra are given in energy, and then stored).
- If an argument `crs_min` is passed, then the entry `crs_exc` from the `settings.py` file is ignored, since this already accounts for an exception.

Importing Mixs

At any time, the function

```
`spcat.mix_out()
```

can be called. This creates two files into the `input/` directory named

```
Natural_out.txt
```

```
Compound_out.txt
```

`Natural_out.txt` is used to create the `Element` instances, but that only needs to be done once and has already been done. `Compound_out.txt` is used to create new `Compound` instances, and can be done at any time. Enter that file, follow the instructions, change the filename to `Compound_in.txt` and run the project (alternatively, run `spcat.mix_in()`). New Compounds will be imported, and their spectra will be stored in the `data/` directory so that this doesn't have to be done every time.

Also, when a file ending in `_in.txt` is imported, the `in` suffix disappears so that it doesn't get imported every time.

Note: The file `Natural.txt` that exists in the root directory of the project is a reminiscence of the time where the `Elements` were created (sounds biblical). It is merely a file that contains the isotope natural abundances of the elements

In depth

Multiple plot

It is possible to plot spectra together by means of:

```
spcat.plot()
```

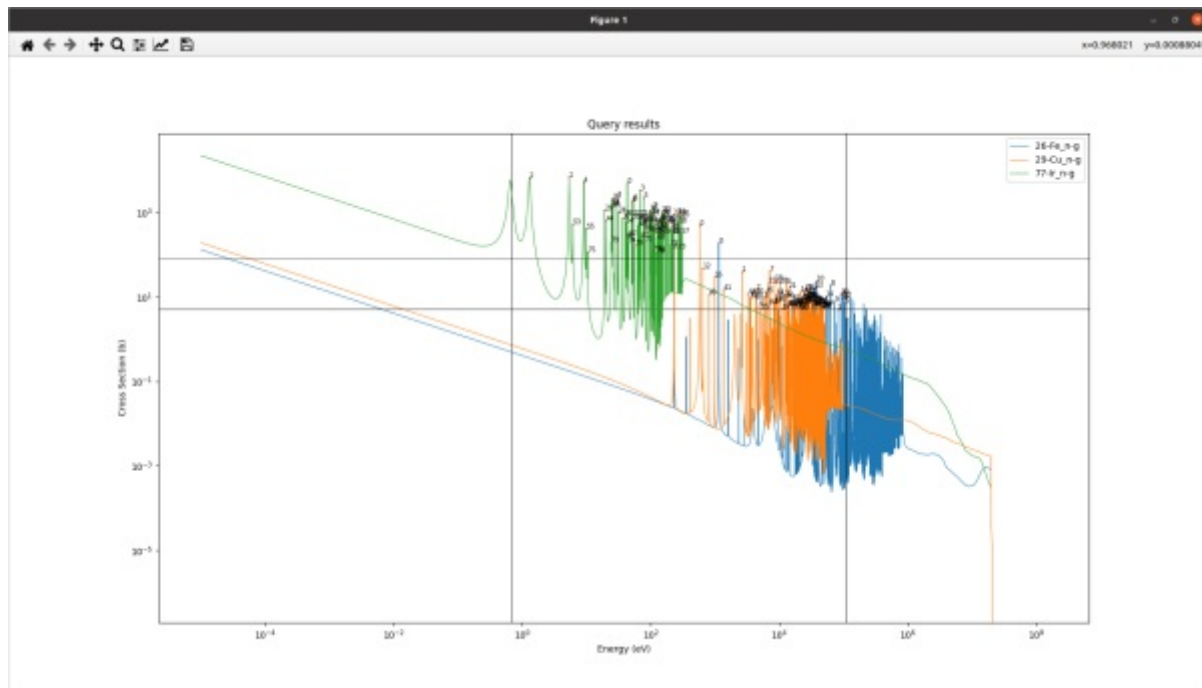
For example:

```

In [2]: spcat.plot()
>Cu
1: 29-Cu-63_n-g
2: 29-Cu-63_n-tot
3: 29-Cu-65_n-g
4: 29-Cu-65_n-tot
5: 29-Cu_n-g
6: 29-Cu_n-tot
Select above results of your query ([0] all; [] None) >5
Current elements: ['29-Cu_n-g']
>Fe
1: 26-Fe-54_n-g
2: 26-Fe-54_n-tot
3: 26-Fe-56_n-g
4: 26-Fe-56_n-tot
5: 26-Fe-57_n-g
6: 26-Fe-57_n-tot
7: 26-Fe-58_n-g
8: 26-Fe-58_n-tot
9: 26-Fe_n-g
10: 26-Fe_n-tot
Select above results of your query ([0] all; [] None) >9
Current elements: ['29-Cu_n-g', '26-Fe_n-g']
>Ir
1: 77-Ir-191_n-g
2: 77-Ir-191_n-tot
3: 77-Ir-193_n-g
4: 77-Ir-193_n-tot
5: 77-Ir_n-g
6: 77-Ir_n-tot
Select above results of your query ([0] all; [] None) >5
Current elements: ['29-Cu_n-g', '26-Fe_n-g', '77-Ir_n-g']
>
x-axis: (1 eV; [2] ToF) >1
Show detection limits? ([y]/n) >y

```

which produces



Bars

Another option is to plot a sample spectrum together with a number of Data instances, which is done with

```
spcat.plotbars()
```

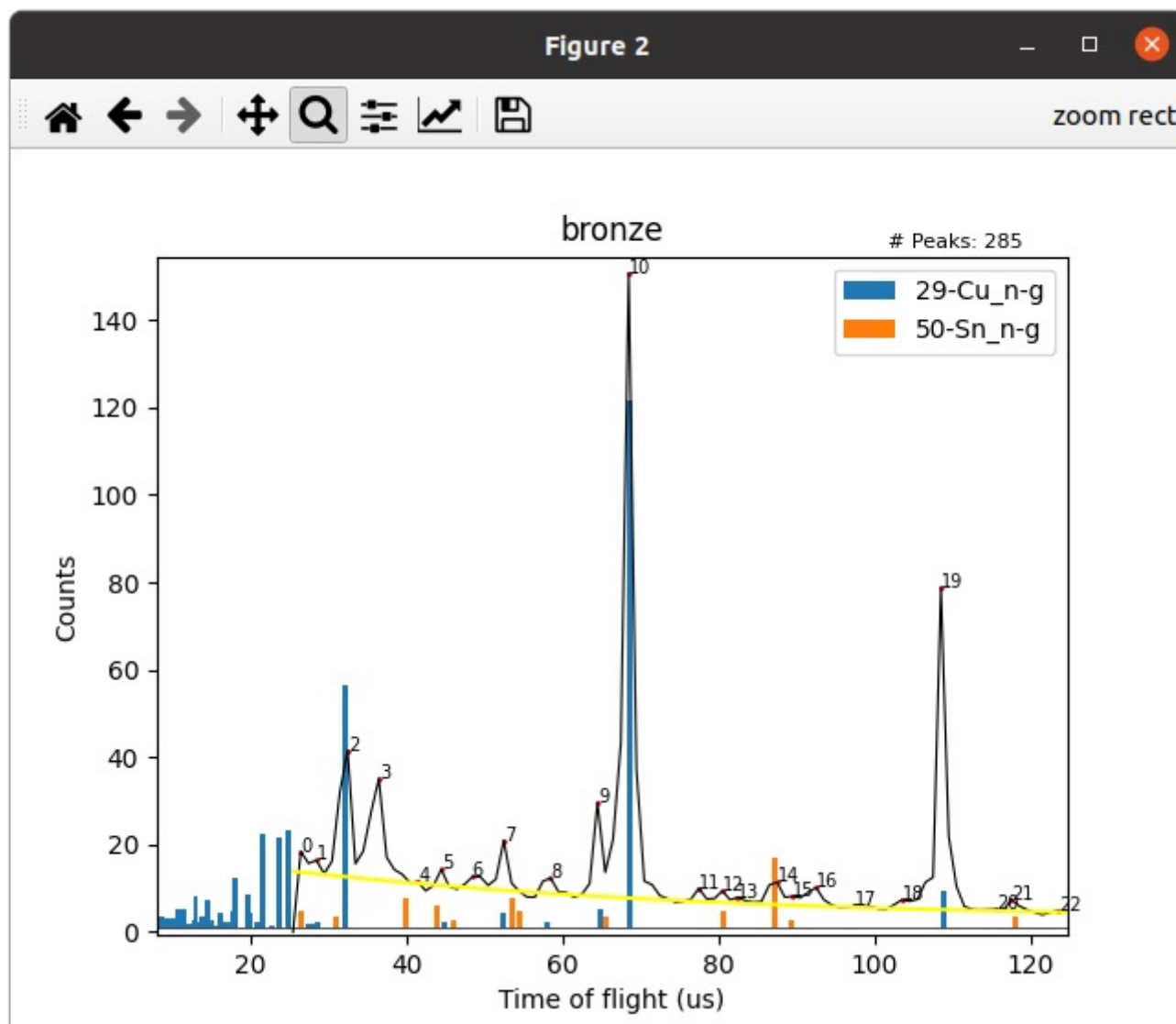
For example:

```

In [3]: spcat.plotbars()
Work on sample:
>bronze
Select peaks from:
>Cu
1: 29-Cu-63_n-g
2: 29-Cu-65_n-g
3: 29-Cu_n-g
Select above results of your query ([0] all; [] None) >3
Current elements: ['29-Cu_n-g']
>Sn
1: 50-Sn-112_n-g
2: 50-Sn-114_n-g
3: 50-Sn-115_n-g
4: 50-Sn-116_n-g
5: 50-Sn-117_n-g
6: 50-Sn-118_n-g
7: 50-Sn-119_n-g
8: 50-Sn-120_n-g
9: 50-Sn-122_n-g
10: 50-Sn-124_n-g
11: 50-Sn_n-g
Select above results of your query ([0] all; [] None) >11
Current elements: ['29-Cu_n-g', '50-Sn_n-g']
>

```

which produces



(zommed-in version)

Here the sample is plotted, the fitted background is shown in yellow, and a bar plot is shown, where the bar heights are (relative to each other within a same compound) proportional to the peak intensity.

Matching

To determine what is there in a sample, the function

```
spcat.pmatch(distmax=cf.max_match, samp=None)
```

can be called, and that ranks by proximity the Data members which have a peak that is close enough to the sample requested one (threshold as `distmax` argument, or else in `Settings.py`). It shows the Data name, the peak label that is close enough, and the distance at which it is to the requested sample peak:

```

In [4]: spcat.pmatch()
>tof1
Enter peak: >68
 0:      63-Eu_n-tot ( 17 - 0.226)
 1:      63-Eu-151_n-tot ( 3 - 0.226)
 2:      63-Eu-153_n-tot ( 42 - 0.566)
 3:      55-Cs-133_n-tot ( 7 - 0.631)
 4:      55-Cs_n-tot ( 7 - 0.631)
 5:      64-Gd-154_n-tot ( 58 - 0.671)
 6:      33-As-75_n-tot ( 32 - 0.855)
 7:      33-As_n-tot ( 32 - 0.855)
 8:      92-U-235_n-tot ( 382 - 0.897)
 9:      64-Gd-155_n-tot ( 83 - 1.044)
10:      64-Gd_n-tot ( 187 - 1.198)
11:      74-W-183_n-tot ( 6 - 1.224)
12:      74-W_n-tot ( 82 - 1.224)
13:      45-Rh-103_n-tot ( 92 - 1.306)
14:      45-Rh_n-tot ( 92 - 1.306)
15:      49-In-115_n-tot ( 90 - 2.142)
16:      49-In_n-tot ( 95 - 2.142)
17:      78-Pt_n-tot ( 205 - 2.181)
18:      78-Pt-192_n-tot ( 5 - 2.181)
19:      56-Ba-130_n-tot ( 5 - 2.718)
Enter peak: >
In [5]: 

```

Comparing

The steps to compare the peaks in order to find the proportions of isotopes/elements or compounds in a sample are the following:

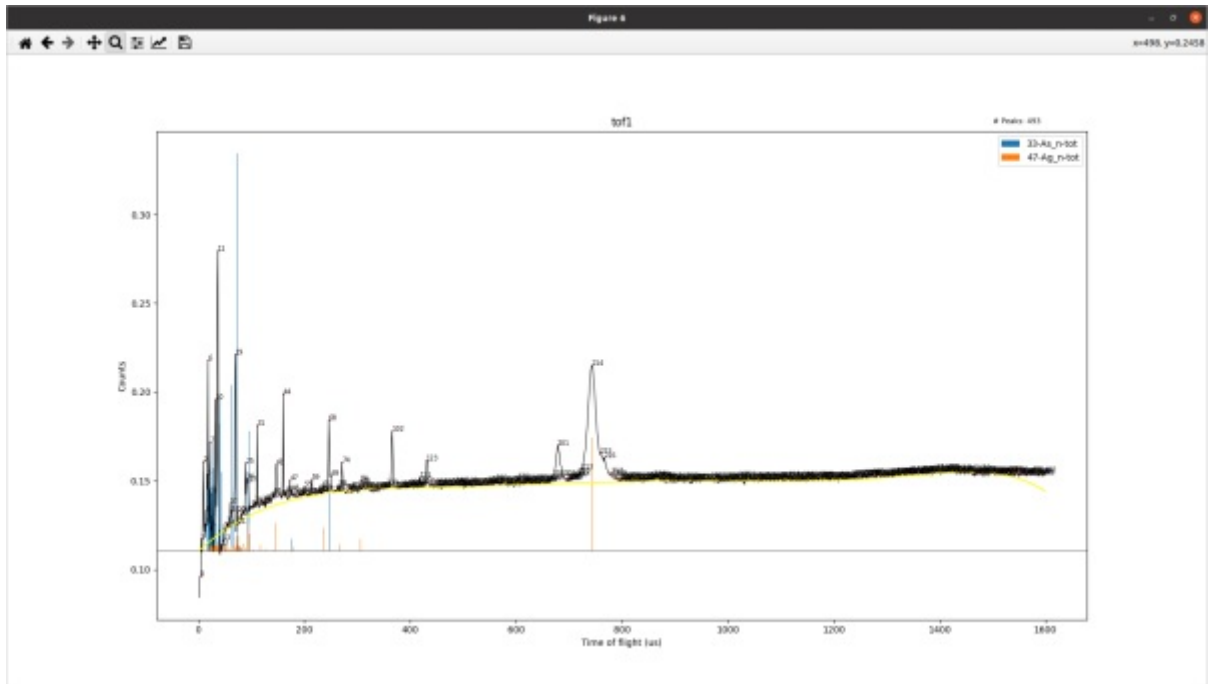
1. Create a variable that will store a Summer object, and name it, say, outcome.
2. Enter the sample to work on
3. Enter the name of one of the Data instances (Isotope, Element or Compound) whose peaks are suspected to be in the sample.
4. Select from the matching results
5. Repeat steps 3 and 4 to include all the desired substances, then hit *Enter*

```

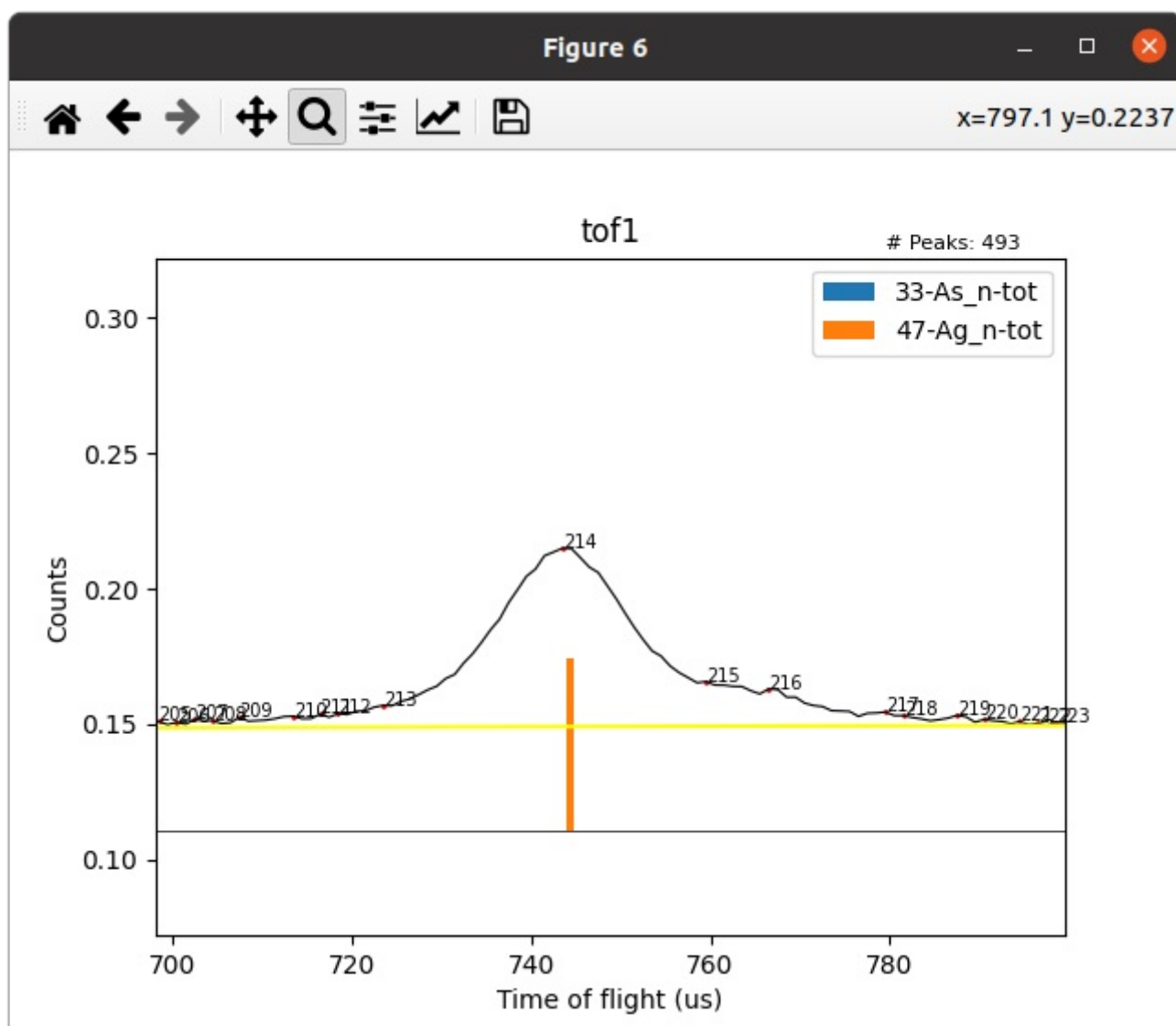
In [9]: outcome = spcat.pcompare()
Work on sample:
>tof1
Select peaks from:
>Ag
1: 47-Ag-107_n-tot
2: 47-Ag-109_n-tot
3: 47-Ag_n-tot
Select above results of your query ([0] all; [] None) >3
Current elements: ['47-Ag_n-tot']
>As
1: 33-As-75_n-tot
2: 33-As_n-tot
Select above results of your query ([0] all; [] None) >2
Current elements: ['47-Ag_n-tot', '33-As_n-tot']
>
Enter peak: >

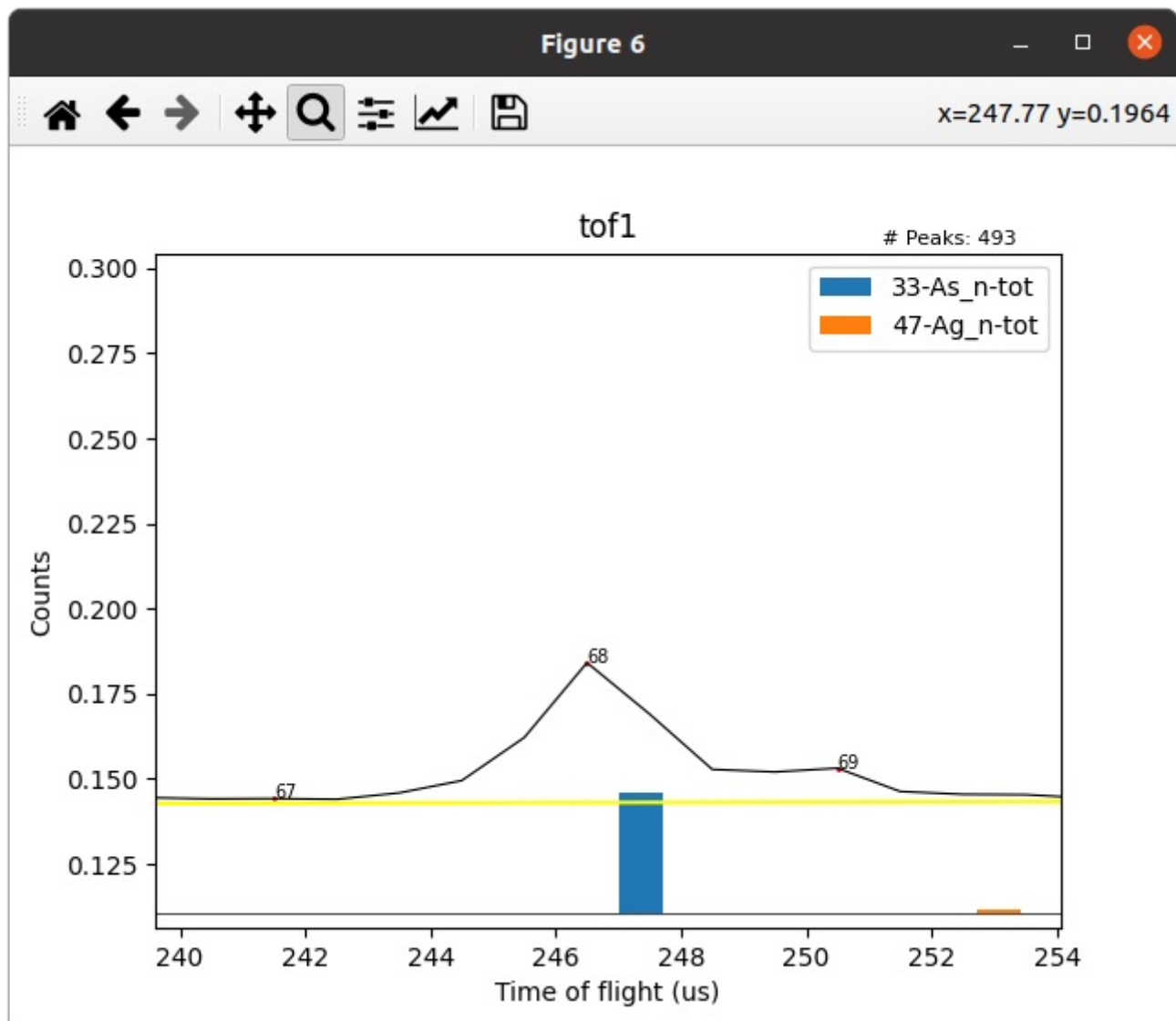
```

A window opens:



In our case, we are interested in the peaks 214 and 68:





6. Enter one of the peaks
7. Enter its boundaries (no spaces, separated by a comma)
8. Select one of the suggested Data elements that have a peak that is close enough.
Note: blank means 0th suggestion.


```

In [9]: outcome = spcat.pcompare()
Work on sample:
>tof1
Select peaks from:
>Ag
1: 47-Ag-107_n-tot
2: 47-Ag-109_n-tot
3: 47-Ag_n-tot
Select above results of your query ([0] all; [] None) >3
Current elements: ['47-Ag_n-tot']
>As
1: 33-As-75_n-tot
2: 33-As_n-tot
Select above results of your query ([0] all; [] None) >2
Current elements: ['47-Ag_n-tot', '33-As_n-tot']
>
Enter peak: >214
    Enter peak boundaries: >700,790
    Peak center: 743.5. Peak newlims: (698, 788)
    Closest component peaks:
    0: 47-Ag_n-tot (0.8652018715177974)
    1: 33-As_n-tot (496.1449324416245)
    Enter one of the above components: >
Closest component: 47-Ag_n-tot. Intensities ratio: 0.000009
Enter peak: >

```

9. Repeat steps 6 to 8 for every one of the sample peaks of interest.
10. An outcome is provided.

```

In [9]: outcome = spcat.pcompare()
Work on sample:
>tof1
Select peaks from:
>Ag
1: 47-Ag-107_n-tot
2: 47-Ag-109_n-tot
3: 47-Ag_n-tot
Select above results of your query ([0] all; [] None) >3
Current elements: ['47-Ag_n-tot']
>As
1: 33-As-75_n-tot
2: 33-As_n-tot
Select above results of your query ([0] all; [] None) >2
Current elements: ['47-Ag_n-tot', '33-As_n-tot']
>
Enter peak: >214
    Enter peak boundaries: >700,790
    Peak center: 743.5. Peak newlims: (698, 788)
    Closest component peaks:
    0: 47-Ag_n-tot (0.8652018715177974)
    1: 33-As_n-tot (496.1449324416245)
    Enter one of the above components: >
Closest component: 47-Ag_n-tot. Intensities ratio: 0.000009
Enter peak: >68
    Enter peak boundaries: >240,250
    Peak center: 246.5. Peak newlims: (238, 248)
    Closest component peaks:
    0: 33-As_n-tot (0.855067558375481)
    1: 47-Ag_n-tot (6.573441280420553)
    Enter one of the above components: >
Closest component: 33-As_n-tot. Intensities ratio: 0.000018
Enter peak: >
Composition results for tof1:
    47-Ag_n-tot: 32.16%
    33-As_n-tot: 67.84%

In [10]: 

```

Now the variable outcome has the information on the result of our investigation:

```

In [12]: outcome.get_as_dict()
Out[12]: {'47-Ag_n-tot': 8.648786456720991e-06, '33-As_n-tot': 1.824555927881831e-05}

In [13]: outcome.get_as_lists()
Out[13]:
(['33-As_n-tot', '47-Ag_n-tot'],
 [1.824555927881831e-05, 8.648786456720991e-06])

In [14]: outcome.get_all()
Out[14]:
{'47-Ag_n-tot': [8.648786456720991e-06],
 '33-As_n-tot': [1.824555927881831e-05]}

In [15]: outcome.get_stats('47-Ag_n-tot')
Out[15]: (8.648786456720991e-06, 0.0)

In [16]: outcome.sum()
Out[16]: 2.6894345735539302e-05

In [17]: outcome.percentage(outof100=True, decimals=4)
Out[17]: {'47-Ag_n-tot': 32.1584, '33-As_n-tot': 67.8416}

```

Note: `outcome.getstats()` returns the **mean** and the **variance** of the intensities ratio for the silver.

Appendix

List of methods and attributes

Catalog

Attributes

- isotopes: dictionary of Isotopes
- elements: dictionary of Elements
- compounds: dictionary of Compounds
- samples: dictionary of Samples
- volumes: a list of the Substance kinds
- date_created
- date_modified

Methods

- `Isotopes(mode=None)`: returns the dictionary of Isotopes. However, if it is called with an argument `n-tot` or `n-g` (`spcat.Isotopes(n-g)`), then it discriminates and choses the desired mode before returning the Isotopes.
- `Elements(mode=None)`: the same with Elements.
- `Compounds(mode=None)`: the same with Compounds
- `Samples(mode=None)`: the same with Samples.
- `Mixes(mode=None)`: the same with Mix classes (see above).
- `Datas(mode=None)`: the same with Data classes (see above).
- `Substances(mode=None)`: the same with Substance classes, namely all of them.
- `get(inp, otherwise=None)`: picks and returns the entry of between all the Substance instances that is named as `inp`. If it doesn't exist returns `otherwise`.

- `find(askmode=cf.ask_mode, ask_if_one=False)`: asks the user a query to look for (e.g. "C" or "C-12") and then select from within the matches. Then, returns the selected item. If `askmode` is set to true, the user will be asked whether to search only from between *n-g* or *n-tot*. Otherwise, all are shown. If not given, it defaults to the settings file. If there is only one match, the user will be asked or not (then simply selected) depending on the value of `ask_if_one`. These two arguments are optional, though.
- `mix_out()`
- `mix_in()`
- `sample_in()`
- `plot()`
- `plotbars()`
- `pcompare()`
- `pmatch()`
- `save()`
- `export()`

Data

Attributes

- `atom`: atomic number, e.g. 26
- `mass`: massic number, e.g. 54
- `symp`: chemical symbol, e.g. Fe
- `mode`: e.g. "n-g"
- `fullname`: e.g. "26-Fe-54_n-g"
- `kind`: "isotope", "element" or "compound"
- `date_created`
- `spectrum`
- `spectrum_tof`
- `der`
- `sder`
- `ma`: maxima
- `mai`: maxima indices
- `ma_tof`: maxima in tof
- `xbounds` energy threshold for peak detection
- `ybounds` XS threshold for peak detection
- `npeaks`: number of Peaks
- `peaks`: **dictionary of Peaks
- `errors`
- `abundances` (only for Mix)
- `components` (only for Mix)

Methods

- `delete()`: delete a peak (interactive)
- `edit()`: edit a peak (interactive)
- `infopeaks()`: prints information on the peaks
- `plot()`
- `plotpeaks()`
- `plotsingle()`
- `recompute(**kwargs)`

Peak

Methods

- `fullname`: Name of the isotope they belong to
- `num`: Peak label, mainly. Should match with `integral_`
- `center_`: Energy rank of peak center. For example: in an isotope with N peaks, 0 is the first peak in the spectrum and N is the last one.
- `center`: Energy value of peak center
- `icenter`: Index number of peak center in raw data file
- `center_tof`: Time of Flight (us) value of peak center
- `integral`: Integral value of peak
- `integral_`: Integral rank of peak. For example: in an isotope with N[0] peaks[errors], 0 is the most intense peak and N is the least intense one.
- `integral_tof`: Integral value of peak (us)
- `width`: Width value of peak (should be `xlims[1]-xlims[0]`)
- `width_`: Width rank of peak
- `height`: Height value of peak. Computed as the XS (or NC) at peak center minus the arithmetic mean of the XSs (or NCs) values on the peak bounds.
- `height_`: Height rank of peak
- `fwhm`: Full width at half maximum of peak, i.e. energy distance when the XS (or NC) has dropped to 1/2 along the peak respect to the maximum value
- `fwhm_`: Full width at half maximum rank of peak
- `ahh`: AHH: Integral divided by squared height
- `ahh_`: AHH rank of peak
- `ahw`: AHW: Integral divided by height times width
- `ahw_`: AHW rank of peak
- `xlims`: Energy values of peak bounds
- `ilims`: Data indices of peak bounds
- `outerslope`: Ideally, slope value far away from the peak and uncertainty (tuple). See documentation in `definepeak` for further information on this.
- `peakreason`: Left and right boundaries reason (tuple). See documentation in `definepeak` for further information on this.
- `yvals`: CS (or NC) values at peak boundaries (tuple)
- `coords`: Energy and CS (or NC) values at peak summit
- `coords_tof`: Time of Flight (us) and CS (or NC) values at peak summit
- `prange`: prange value. See documentation in `definepeak` for further information on this.
- `user_edited`: True if user has edited the peak, False if the peak attributes are all from computation.
- `user_defined`: True if user has defined the peak, False if the peak is defined from computation.

Note: **CS** stands for *Cross-Section* and **NC** for *Number of Counts*.

Arrays

Arrays with x,y values are considered to be horizontal, e.g.

```
[ [ x0  x1  x2  x3  ... ]
  [ y1  y2  y3  y4  ... ] ]
```

This means:

- `arr[0]`: x-values
- `arr[1]`: y-values
- `arr[:,0]`: 0th x-y pair

And so on.