
PyMassSpec

Release 2.3.0

Python Toolkit for Mass Spectrometry

PyMassSpec Authors

Sep 09, 2022

Contents

1	The PyMassSpec project	3
2	Features	5
3	Installation	7
3.1	from PyPI	7
3.2	from Anaconda	7
3.3	from GitHub	7
4	Usage	9
4.1	Example processing GC-MS data	9
5	License	11
6	Issues	13
6.1	Installation	13
6.2	User Guide	13
6.3	Documentation	65
6.4	Demos and Examples	161
6.5	Contributing	173
	Python Module Index	183
	Index	185

PyMassSpec is a [Python](#) package for processing gas chromatography-mass spectrometry data. PyMassSpec provides a framework and a set of components for rapid development and testing of methods for processing of chromatography-mass spectrometry data. PyMassSpec can be used interactively through the Python shell, in a [Jupyter Notebook](#), or the functions can be collected into scripts when it is preferable to perform data processing in the batch mode.

Forked from the original PyMS Repository: <https://github.com/ma-bio21/pyms>. Originally by Andrew Isaac, Sean O'Callaghan and Vladimir Likić. The original publication can be found here: <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-13-115>

The original project seems to have been abandoned as there has been no activity since 2017.

Table of Contents

- *The PyMassSpec project*
- *Features*
- *Installation*
 - *from PyPI*
 - *from Anaconda*
 - *from GitHub*
- *Usage*
 - *Example processing GC-MS data*
- *License*
- *Issues*
 - *Installation*
 - * *from PyPI*
 - * *from Anaconda*
 - * *from GitHub*

The PyMassSpec project

The directory structure of PyMassSpec is as follows:

```
/
├── pyms:      The PyMassSpec code
├── pyms-data: Example GC-MS data files
├── pyms-demo: Examples of how to use PyMassSpec
├── tests:     pytest tests
└── doc-source: Sphinx source for documentation
```


Features

Installation

3.1 from PyPI

```
$ python3 -m pip install PyMassSpec --user
```

3.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/bioconda
$ conda config --add channels https://conda.anaconda.org/conda-forge
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install PyMassSpec
```

3.3 from GitHub

```
$ python3 -m pip install git+https://github.com/PyMassSpec/PyMassSpec@master --user
```


Usage

A tutorial illustrating various PyMassSpec features in detail is provided in subsequent chapters of this User Guide. The commands executed interactively are grouped together by example, and can be found [here](#).

The data used in the PyMassSpec documentation and examples is available [here](#).

In the “*Demos and Examples*” section there is a page corresponding to each example, coded with the chapter number (ie. “pyms-demo/20a/” corresponds to the Example 20a, from Chapter 2).

Each example has a script named ‘proc.py’ which contains the commands given in the example. These scripts can be run with the following command:

```
$ python3 proc.py
```

4.1 Example processing GC-MS data

Download the file `gc01_0812_066.jdx` and save it in the folder `data`. This file contains GC-MS data in the the JCAMP-DX format.

First the raw data is loaded:

```
>>> from pyms.GCMS.IO.JCAMP import JCAMP_reader
>>> jcamp_file = "data/gc01_0812_066.jdx"
>>> data = JCAMP_reader(jcamp_file)
-> Reading JCAMP file 'Data/gc01_0812_066.jdx'
>>> data
<pyms.GCMS.Class.GCMS_data at 0x7f3ec77da0b8>
```

The intensity matrix object is then built by binning the data:

```
>>> from pyms.IntensityMatrix import build_intensity_matrix_i
>>> im = build_intensity_matrix_i(data)
```

In this example, we show how to obtain the dimensions of the newly created intensity matrix, then loop over all ion chromatograms, and for each ion chromatogram apply Savitzky-Golay noise filter and tophat baseline correction:

```
>>> n_scan, n_mz = im.size
>>> from pyms.Noise.SavitzkyGolay import savitzky_golay
>>> from pyms.TopHat import tophat
>>> for ii in range(n_mz):
...     print("working on IC", ii)
...     ic = im.get_ic_at_index(ii)
...     ic1 = savitzky_golay(ic)
...     ic_smooth = savitzky_golay(ic1)
...     ic_base = tophat(ic_smooth, struct="1.5m")
...     im.set_ic_at_index(ii, ic_base)
```

The resulting noise and baseline corrected ion chromatogram is saved back into the intensity matrix.

Further examples can be found in the [documentation](#)

License

PyMassSpec is Free and Open Source software released under the [GNU General Public License version 2](#).

Issues

If you encounter any problems, please [file an issue](#) along with a detailed description.

6.1 Installation

6.1.1 from PyPI

```
$ python3 -m pip install PyMassSpec --user
```

6.1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/bioconda
$ conda config --add channels https://conda.anaconda.org/conda-forge
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install PyMassSpec
```

6.1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/PyMassSpec/PyMassSpec@master --user
```

6.2 User Guide

6.2.1 GC-MS Raw Data Model

Table of Contents

- *Introduction*
- *Example: Reading JCAMP GC-MS data*
 - *A GCMS_data Object*
 - *A Scan Object*
 - *Exporting data and obtaining information about a data set*
- *Example: Comparing two GC-MS data sets*

Introduction

PyMassSpec can read gas chromatography-mass spectrometry (GC-MS) data stored in Analytical Data Interchange for Mass Spectrometry (ANDI-MS),¹ and Joint Committee on Atomic and Molecular Physical Data (JCAMP-DX)² formats. The information contained in the data files can vary significantly depending on the instrument, vendor's software, or conversion utility. PyMassSpec makes the following assumptions about the information contained in the data file:

- The data contain the m/z and intensity value pairs across a scan.
- Each scan has a retention time.

Internally, PyMassSpec stores the raw data from ANDI files or JCAMP files as a `GCMS_data` object.

Example: Reading JCAMP GC-MS data

The PyMS package `pyms.GCMS.IO.JCAMP` provides capabilities to read the raw GC-MS data stored in the JCAMP-DX format.

First, setup the paths to the datafile and the output directory, then import JCAMP_reader.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.GCMS.IO.JCAMP import JCAMP_reader
```

Read the raw JCAMP-dx data.

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
data
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

```
<GCMS_data(305.582 - 4007.722 seconds, time step 0.3753183292781833, 9865 scans)>
```

A GCMS_data Object

The object data (from the two previous examples) stores the raw data as a `pyms.GCMS.Class.GCMS_data` object. Within the `GCMS_data` object, raw data are stored as a list of `pyms.Spectrum.Scan` objects and a list of retention times. There are several methods available to access data and attributes of the `GCMS_data` and `Scan` objects.

The `GCMS_data` object's methods relate to the raw data. The main properties relate to the masses, retention times and scans. For example, the minimum and maximum mass from all of the raw data can be returned by the following:

```
In [3]: data.min_mass
```

```
50.0
```

¹ ANDI-MS was developed by the Analytical Instrument Association

² JCAMP-DX is maintained by the International Union of Pure and Applied Chemistry

```
In [4]: data.max_mass
```

```
599.9
```

A list of the first 10 retention times can be returned with:

```
In [5]: data.time_list[:10]
```

```
[305.582,
 305.958,
 306.333,
 306.708,
 307.084,
 307.459,
 307.834,
 308.21,
 308.585,
 308.96]
```

The index of a specific retention time (in seconds) can be returned with:

```
In [6]: data.get_index_at_time(400.0)
```

```
252
```

Note that this returns the index of the retention time in the data closest to the given retention time of 400.0 seconds.

The `GCMS_data.tic` attribute returns a total ion chromatogram (TIC) of the data as an *IonChromatogram* object:

```
In [7]: data.tic
```

```
<pyms.IonChromatogram.IonChromatogram at 0x7f6b22ff9d68>
```

The *IonChromatogram* object is explained in a later example.

A Scan Object

A `pyms.Spectrum.Scan` object contains a list of masses and a corresponding list of intensity values from a single mass-spectrum scan in the raw data. Typically only non-zero (or non-threshold) intensities and corresponding masses are stored in the raw data.

A list of the first 10 `pyms.Spectrum.Scan` objects can be returned with:

```
In [8]: scans = data.scan_list
        scans[:10]
```

```
[<pyms.Spectrum.Scan at 0x7f6b4117a518>,
 <pyms.Spectrum.Scan at 0x7f6b22ff9400>,
 <pyms.Spectrum.Scan at 0x7f6b22ff9dd8>,
 <pyms.Spectrum.Scan at 0x7f6b22ff9e80>,
 <pyms.Spectrum.Scan at 0x7f6b22ff9f28>,
 <pyms.Spectrum.Scan at 0x7f6b22ff9fd0>,
 <pyms.Spectrum.Scan at 0x7f6b22ff9e48>,
```

(continues on next page)

(continued from previous page)

```
<pyms.Spectrum.Scan at 0x7f6b22ff9668>,
<pyms.Spectrum.Scan at 0x7f6b22ff9d30>,
<pyms.Spectrum.Scan at 0x7f6b22ff9cf8>]
```

A list of the first 10 masses in a scan (e.g. the 1st scan) is returned with:

```
In [9]: scans[0].mass_list[:10]
```

```
[50.1, 51.1, 53.1, 54.2, 55.1, 56.2, 57.2, 58.2, 59.1, 60.1]
```

A list of the first 10 corresponding intensities in a scan is returned with:

```
In [10]: scans[0].intensity_list[:10]
```

```
[22128.0,
 10221.0,
 31400.0,
 27352.0,
 65688.0,
 55416.0,
 75192.0,
 112688.0,
 152256.0,
 21896.0]
```

The minimum and maximum mass in an individual scan (e.g. the 1st scan) are returned with:

```
In [11]: scans[0].min_mass
```

```
50.1
```

```
In [12]: scans[0].max_mass
```

```
599.4
```

Exporting data and obtaining information about a data set

Often it is of interest to find out some basic information about the data set, e.g. the number of scans, the retention time range, and m/z range and so on. The `GCMS_data` class provides a method `info()` that can be used for this purpose.

```
In [13]: data.info()
```

```
Data retention time range: 5.093 min -- 66.795 min
Time step: 0.375 s (std=0.000 s)
Number of scans: 9865
Minimum m/z measured: 50.000
Maximum m/z measured: 599.900
Mean number of m/z values per scan: 56
Median number of m/z values per scan: 40
```

The entire raw data of a `GCMS_data` object can be exported to a file with the method `write()`:

```
In [14]: data.write(output_directory / "data")
```

```
-> Writing intensities to '/home/vagrant/PyMassSpec/pyms-demo/jupyter/output/data.I.
    ↳ csv'
-> Writing m/z values to '/home/vagrant/PyMassSpec/pyms-demo/jupyter/output/data.mz.
    ↳ csv'
```

This method takes the filename (“output/data”, in this example) and writes two CSV files. One has extension “.I.csv” and contains the intensities (“output/data.I.csv” in this example), and the other has the extension “.mz” and contains the corresponding table of m/z value (“output/data.mz.csv” in this example). In general, these are not two-dimensional matrices, because different scans may have different number of m/z values recorded.

Note: This example is in `pyms-demo/jupyter/reading_jcamp.ipynb`. There is also an example in that directory for reading ANDI-MS files.

Example: Comparing two GC-MS data sets

Occasionally it is useful to compare two data sets. For example, one may want to check the consistency between the data set exported in netCDF format from the manufacturer’s software, and the JCAMP format exported from a third party software.

First, setup the paths to the datafiles and the output directory, then import JCAMP_reader and ANDI_reader.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.GCMS.IO.JCAMP import JCAMP_reader
from pyms.GCMS.IO.ANDI import ANDI_reader
```

Then the raw data is read as before.

```
In [2]: andi_file = data_directory / "gc01_0812_066.cdf"
data1 = ANDI_reader(andi_file)
data1
```

```
-> Reading netCDF file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.cdf'
```

```
<GCMS_data(305.582 - 4007.721 seconds, time step 0.37531822789943226, 9865 scans)>
```

```
In [3]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data2 = JCAMP_reader(jcamp_file)
data2
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

```
<GCMS_data(305.582 - 4007.722 seconds, time step 0.3753183292781833, 9865 scans)>
```

To compare the two data sets, use the function `diff()`

```
In [4]: from pyms.GCMS.Function import diff
```

```
diff(data1, data2)
```

```
Data sets have the same number of time points.  
Time RMSD: 3.54e-04  
Checking for consistency in scan lengths ...OK  
Calculating maximum RMSD for m/z values and intensities ...  
Max m/z RMSD: 1.03e-05  
Max intensity RMSD: 0.00e+00
```

If the data cannot be compared, for example because of different number of scans, or inconsistent number of m/z values in between two scans, `diff()` will report the difference. For example:

```
In [5]: data2.trim(begin=1000, end=2000)
```

```
Trimming data to between 1000 and 2001 scans
```

```
In [6]: diff(data1, data2)
```

```
The number of retention time points differ.  
First data set: 9865 time points  
Second data set: 1002 time points  
Data sets are different.
```

Note: This example is in `pyms-demo/jupyter/comparing_datasets.ipynb`.

6.2.2 GC-MS data derived objects

Table of Contents

- *IntensityMatrix Object*
 - *Discussion of Binning Boundaries*
 - *Example: Building an Intensity Matrix*
 - *Build intensity matrix parameters*
 - *Build integer mass intensity matrix*
- *Example: MassSpectrum Objects*
- *Example: IonChromatogram Objects*
 - *Writing IonChromatogram object to a file*
- *Saving data*
- *Importing ASCII data*

In the raw GC-MS data, consecutive scans do not necessarily contain the same mass per charge (mass) values. For data processing, it is often necessary to convert the data to a matrix with a set number of masses and scans. In

PyMassSpec the resulting object is called an intensity matrix. In this chapter the methods for converting the raw GC-MS data to an intensity matrix object are illustrated.

IntensityMatrix Object

The general scheme for converting raw mass values is to bin intensity values based on the interval the corresponding mass belongs to. The general procedure is as follows:

- Set the interval between bins, lower and upper bin boundaries.
- Calculate the number of bins to cover the range of all masses.
- Centre the first bin at the minimum mass found for all the raw data.
- Sum intensities whose masses are in a given bin.

A mass, m , is considered to belong to a bin when $c - l \leq m < c + u$, where c is the centre of the bin, l is the lower boundary and u is the upper boundary of the bin. The default bin interval is one with a lower and upper boundary of ± 0.5 .

A function to bin masses to the nearest integer is also available. The default bin interval is one with a lower boundary of -0.3 and upper boundary of $+0.7$ (as per the NIST library).

Discussion of Binning Boundaries

For any chemical element X , let $w(x)$ be the atomic weight of X , and

$$\delta(X) = \frac{w(X) - \{w(X)\}}{w(X)}$$

where $\{a\}$ is the integer value of a (rounded to the nearest integer).

For example, for hydrogen $\delta(^1\text{H}) = \frac{1.007825032 - 1}{1.007825032} = 0.0076$. Similarly $\delta(^{12}\text{C}) = 0$, $\delta(^{14}\text{N}) = 0.00022$, $\delta(^{16}\text{O}) = -0.00032$, etc.

Let also $\Delta(X) = w(X) - \{w(X)\}$. Then $-0.023 < \Delta(^{31}\text{P}), \Delta(^{28}\text{Si}) < 0$.

Let a compound undergo GC-MS and let Y be one of its fragments. If Y consists of k_1, k_2 atoms of type X_2, \dots, k_r atoms of type X_r , then $\Delta(Y) = k_1 * \Delta(X_1) + k_2 * \Delta(X_2) + \dots + k_r * \Delta(X_r)$.

The fragment will usually not contain more than 2 or 3 P or Si atoms and if its molecular weight is less than 550 it may not contain more than 35 O atoms, so $\Delta(Y) \geq -0.023 * 5 - 0.00051 * 35 = -0.133$.

On the other hand, if Y contains k H atoms and m N atoms, then $\Delta(Y) \leq k * 0.00783 + m * 0.00051$. Since for each two hydrogen atoms at least one carbon (or heavier) atom is needed, giving the limit of no more than 80 hydrogen atoms. Therefore in this case (i.e. H and C atoms only) $\Delta(Y) \leq 80 * 0.00783 = 0.63$. If carbon is replaced by any heavier atom, at least 2 hydrogen atoms will be eliminated and $\Delta(Y)$ will become even smaller.

If the molecular weight of Y does not exceed 550 (typically the largest mass scanned for in a GC-MS setup) then $-0.133 \leq \Delta(Y) \leq 0.63$. This means that if we set our binning boundaries to $(-0.3, 0.7)$ or $(-0.2, 0.8)$ the opportunity for having a fragment whose molecular weight is very close to the boundary is minimised.

Since the resolution of MS is at least 0.1 dalton, we may assume that its error does not exceed 0.05, and MS accuracy will not cause additional problems.

Example: Building an Intensity Matrix

First, setup the paths to the datafiles and the output directory, then import JCAMP_reader.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pym-s-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pym.s.GCMS.IO.JCAMP import JCAMP_reader
```

Read the raw data files.

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
data
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pym-s-data/gc01_0812_066.jdx'
```

```
<GCMS_data(305.582 - 4007.722 seconds, time step 0.3753183292781833, 9865 scans)>
```

Then the data can be converted to an *IntensityMatrix* using the function *build_intensity_matrix()* from *pym.s.IntensityMatrix*.

The default operation of *build_intensity_matrix()* is to use a bin interval of one and treat the masses as floating point numbers. The default intensity matrix can be built as follows:

```
In [3]: from pym.s.IntensityMatrix import build_intensity_matrix

im = build_intensity_matrix(data)

im
```

```
<pym.s.IntensityMatrix.IntensityMatrix at 0x7f31d8b12860>
```

The size as the number of scans and the number of bins can be returned with:

```
In [4]: im.size
```

```
(9865, 551)
```

There are 9865 scans and 551 bins in this example.

The raw masses have been binned into new mass units based on the minimum mass in the raw data and the bin size. A list of the first ten new masses can be obtained as follows:

```
In [5]: im.mass_list[:10]
```

```
[50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0]
```

The attributes *im.min_mass* and *im.max_mass* return the minimum and maximum mass:

```
In [6]: im.min_mass
```



```
50.0
```

```
In [7]: im.max_mass
```

```
600.0
```

It is also possible to search for a particular mass, by finding the index of the binned mass closest to the desired mass. For example, the index of the closest binned mass to a mass of 73.3 m/z can be found by using the methods `im.get_index_of_mass()`:

```
In [8]: index = im.get_index_of_mass(73.3)
```

```
index
```

```
23
```

The value of the closest mass can be returned by the method `im.get_mass_at_index()`:

```
In [9]: im.get_mass_at_index(index)
```

```
73.0
```

A mass of 73.0 is returned in this example.

Build intensity matrix parameters

The bin interval can be set to values other than one, and binning boundaries can also be adjusted. In the example below, to fit the 0.5 bin interval, the upper and lower boundaries are set to ± 0.25 .

```
In [10]: im = build_intensity_matrix(data, 0.5, 0.25, 0.25)
```

```
im
```

```
<pyms.IntensityMatrix.IntensityMatrix at 0x7f31d8b8d710>
```

The size of the intensity matrix will reflect the change in the number of bins:

```
In [11]: im.size
```

```
(9865, 1101)
```

```
In [12]: im.mass_list[:10]
```

```
[50.0, 50.5, 51.0, 51.5, 52.0, 52.5, 53.0, 53.5, 54.0, 54.5]
```

In this example there are 9865 scans (as before), but 1101 bins.

The index and binned mass of the mass closest to 73.3 should also reflect the different binning.

```
In [13]: index = im.get_index_of_mass(73.3)
```

```
index
```

```
47
```

```
In [14]: im.get_mass_at_index(index)
```

```
73.5
```

Build integer mass intensity matrix

It is also possible to build an intensity matrix with integer masses and a bin interval of one using `build_intensity_matrix_i()`. The default range for the binning is -0.3 and +0.7 mass units. The function is imported from `pyms.IntensityMatrix`:

```
In [15]: from pyms.IntensityMatrix import build_intensity_matrix_i

im = build_intensity_matrix_i(data)
im
```

```
<pyms.IntensityMatrix.IntensityMatrix at 0x7f31d8b121d0>
```

```
In [16]: im.size
```

```
(9865, 551)
```

```
In [17]: im.mass_list[:10]
```

```
[50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
```

The masses are now integers.

```
In [18]: index = im.get_index_of_mass(73.3)
index
```

```
23
```

```
In [19]: im.get_mass_at_index(index)
```

```
73
```

The lower and upper bounds can be adjusted with `build_intensity_matrix_i(data, lower, upper)`.

Note: This example is in `pyms-demo/jupyter/IntensityMatrix.ipynb`.

Example: MassSpectrum Objects

First, setup the paths to the datafiles and the output directory, then import JCAMP_reader and build_intensity_matrix.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pym-s-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pym.GCMS.IO.JCAMP import JCAMP_reader
from pym.IntensityMatrix import build_intensity_matrix
```

Read the raw data files and create the IntensityMatrix.

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
im = build_intensity_matrix(data)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pym-s-data/gc01_0812_066.jdx'
```

A *MassSpectrum* object contains two attributes, *mass_list* and *intensity_list*, a list of mass values and corresponding intensities, respectively. A *MassSpectrum* is returned by the *IntensityMatrix* method *get_ms_at_index(index)*.

For example, the properties of the first *MassSpectrum* object can be obtained as follows:

```
In [3]: ms = im.get_ms_at_index(0)

ms
```

```
<pym.Spectrum.MassSpectrum at 0x7ff678cfe080>
```

```
In [4]: len(ms)
```

```
551
```

```
In [5]: len(ms.mass_list)
```

```
551
```

```
In [6]: len(ms.intensity_list)
```

```
551
```

The length of all attributes should be the same.

Note: This example is in `pym-demo/jupyter/MassSpectrum.ipynb`.

Example: IonChromatogram Objects

First, setup the paths to the datafiles and the output directory, then import JCAMP_reader and build_intensity_matrix.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.GCMS.IO.JCAMP import JCAMP_reader
from pyms.IntensityMatrix import build_intensity_matrix
```

Read the raw data files and create the IntensityMatrix.

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
im = build_intensity_matrix(data)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

An *IonChromatogram* object is a one dimensional vector containing mass intensities as a function of retention time. This can be either m/z channel intensities (for example, the ion chromatogram at 73 m/z), or cumulative intensities over all measured m/z (TIC).

An *IonChromatogram* object for the TIC can be obtained as follows:

```
In [3]: data.tic
```

```
<pyms.IonChromatogram.IonChromatogram at 0x7f698cbb9e80>
```

The *IonChromatogram* at index 0 can be obtained with:

```
In [4]: im.get_ic_at_index(0)
```

```
<pyms.IonChromatogram.IonChromatogram at 0x7f69ac4e9198>
```

The *IonChromatogram* for the closest mass to 73 can be obtained with:

```
In [5]: im.get_ic_at_mass(73)
```

```
<pyms.IonChromatogram.IonChromatogram at 0x7f69ac4e95f8>
```

An ion chromatogram object has a method *is_tic()* which returns True if the ion chromatogram is a TIC, False otherwise.

```
In [6]: data.tic.is_tic()
```

```
True
```

```
In [7]: im.get_ic_at_mass(73).is_tic()
```

```
False
```

Note: This example is in `pyms-demo/jupyter/IonChromatogram.ipynb`.

Writing IonChromatogram object to a file

Note: This example is in `pyms-demo/31`

The method `write()` of an `IonChromatogram` object allows the ion chromatogram to be saved to a file:

```
>>> tic.write("output/tic.dat", minutes=True)
>>> im.get_ic_at_mass(73).write("output/ic.dat", minutes=True)
```

The flag `minutes=True` indicates that retention time will be saved in minutes. The ion chromatogram object saved with the `write()` method is a plain ASCII file which contains a pair of (retention time, intensity) per line.

```
$ head tic.dat
5.0930 2.222021e+07
5.0993 2.212489e+07
5.1056 2.208650e+07
5.1118 2.208815e+07
5.1181 2.200635e+07
5.1243 2.200326e+07
5.1306 2.202363e+07
5.1368 2.198357e+07
5.1431 2.197408e+07
5.1493 2.193351e+07
```

Saving data

Note: This example is in `pyms-demo/32`

A matrix of intensity values can be saved to a file with the function `save_data()` from `pyms.Utills.IO`. A matrix of intensity values can be returned from an `IntensityMatrix` with the method `intensity_array`. For example,

```
>>> from pyms.Utills.IO import save_data
>>> mat = im.intensity_array
array([[22128.,    0., 10221., ...,    0.,   470.,    0.],
       [22040.,    0., 10335., ...,   408.,    0.,   404.],
       [21320.,    0., 10133., ...,   492.,    0.,   422.],
       ...,
       [    0.,    0.,    0., ...,    0.,    0.,    0.],
       [    0.,    0.,    0., ...,    0.,    0.,    0.],
       [    0.,    0.,    0., ...,    0.,    0.,    0.]])
>>> save_data("output/im.dat", mat)
```

It is also possible to save the list of masses (from `im.mass_list` and the list of retention times (from `im.time_list`) using the `save_data()` function. For convenience, the intensity values, mass list and time list, can be saved with the method `export_ascii()`. For example,

```
>>> im.export_ascii("output/data")
```

will create `data.im.dat`, `data.rt.dat` and `data.mz.dat`, where these are the intensity matrix, retention time vector, and m/z vector. By default the data is saved as space separated data with a “.dat” extension. It is also possible to save the data as comma separated data with a “.csv” extension with the command:

```
>>> im.export_ascii("output/data", "csv")
```

Additionally, the entire *IntensityMatrix* can be exported to LECO CSV format.

```
>>> im.export_leco_csv("output/data_leco.csv")
```

This facility is useful for import into other analytical software packages. The format has a header line specifying the column heading information as:

```
scan, retention time, mass1, mass2, ...
```

and then each row as the intensity data.

Importing ASCII data

Note: This example is in *pymS-demo/32*

The LECO CSV format data can be imported directly into an *IntensityMatrix* object. The data must follow the format outlined above. For example, the file saved above can be read and compared to the original:

```
>>> from pymS.IntensityMatrix import IntensityMatrix
>>> iim = IntensityMatrix([0],[0],[[0]])
>>> iim.import_leco_csv("output/data_leco.csv")
>>> im.size
>>> iim.size
```

The line `IntensityMatrix([0],[0],[[0]])` is required to create an empty *IntensityMatrix* object.

6.2.3 Data Filtering

Table of Contents

- *Data Filtering*
 - *Introduction*
 - *Time strings*
 - *Example: IntensityMatrix Resizing*
 - * *Retention time range*
 - * *Mass Spectrum range and entries*
 - *Noise smoothing*
 - * *Window averaging*

- * *Window Averaging on Intensity Matrix*
- * *Savitzky–Golay noise filter*
 - *Savitzky-Golay Noise filtering of Intensity Matrix Object*
- *Baseline Correction*
 - * *Tophat Baseline correction on an Intensity Matrix object*
- *Pre-processing the IntensityMatrix*
- *References*

Introduction

In this chapter filtering techniques that allow pre-processing of GC-MS data for analysis and comparison to other pre-processed GC-MS data are covered.

Time strings

Before considering the filtering techniques, the mechanism for representing retention times is outlined here.

A time string is the specification of a time interval, that takes the format `NUMBERs` or `NUMBERm` for time interval in seconds or minutes. For example, these are valid time strings: `10s` (10 seconds) and `0.2m` (0.2 minutes).

Example: IntensityMatrix Resizing

Once an `IntensityMatrix` has been constructed from the raw GC-MS data, the entries of the matrix can be modified. These modifications can operate on the entire matrix, or individual masses or scans.

First, setup the paths to the datafiles and the output directory, then import `JCAMP_reader` and `build_intensity_matrix`.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pym-s-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pym.s.GCMS.IO.JCAMP import JCAMP_reader
from pym.s.IntensityMatrix import build_intensity_matrix
```

Read the raw data files and create the `IntensityMatrix`.

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
im = build_intensity_matrix(data)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pym-s-data/gc01_0812_066.jdx'
```

Retention time range

A basic operation on the GC-MS data is to select a specific time range for processing. In PyMassSpec, any data outside the chosen time range is discarded. The `trim()` method operates on the raw data, so any subsequent processing only refers to the trimmed data.

The data can be trimmed to specific scans:

```
In [3]: data.trim(1000, 2000)
data.info()
```

```
Trimming data to between 1000 and 2001 scans
Data retention time range: 11.342 min -- 17.604 min
Time step: 0.375 s (std=0.000 s)
Number of scans: 1002
Minimum m/z measured: 50.100
Maximum m/z measured: 467.100
Mean number of m/z values per scan: 57
Median number of m/z values per scan: 44
```

or specific retention times (in seconds or minutes):

```
In [4]: data.trim("700s", "15m")
data.info()
```

```
Trimming data to between 54 and 587 scans
Data retention time range: 11.674 min -- 15.008 min
Time step: 0.375 s (std=0.000 s)
Number of scans: 534
Minimum m/z measured: 50.100
Maximum m/z measured: 395.200
Mean number of m/z values per scan: 59
Median number of m/z values per scan: 47
```

Mass Spectrum range and entries

An *IntensityMatrix* object has a set mass range and interval that is derived from the data at the time of building the intensity matrix. The range of mass values can be cropped. This is done, primarily, to ensure that the range of masses used are consistent when comparing samples.

The mass range of the intensity matrix can be “cropped” to a new (smaller) range as follows:

```
In [5]: im.crop_mass(60, 400)
im.min_mass
```

```
60.0
```

```
In [6]: im.max_mass
```

```
400.0
```

It is also possible to set all intensities for a given mass to zero. This is useful for ignoring masses associated with sample preparation. The mass can be “nulled” with:


```
In [7]: im.null_mass(73)
sum(im.get_ic_at_mass(73).intensity_array)
```

```
0.0
```

As expected, the sum of the intensity array is 0

Note: This example is in `pyms-demo/jupyter/IntensityMatrix_Resizing.ipynb`.

Noise smoothing

The purpose of noise smoothing is to remove high-frequency noise from data, and thereby increase the contribution of the signal relative to the contribution of the noise.

First, setup the paths to the datafiles and the output directory, then import JCAMP_reader.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.GCMS.IO.JCAMP import JCAMP_reader
```

Read the raw data files and extract the TIC.

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
tic = data.tic
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

Window averaging

A simple approach to noise smoothing is moving average window smoothing. In this approach the window of a fixed size ($2N+1$ points) is moved across the ion chromatogram, and the intensity value at each point is replaced with the mean intensity calculated over the window size. The example below illustrates smoothing of TIC by window averaging.

To apply mean window smoothing with a 5-point window:

```
In [3]: from pyms.Noise.Window import window_smooth
tic1 = window_smooth(tic, window=5)
```

To apply median window smoothing with a 5-point window:

```
In [4]: tic2 = window_smooth(tic, window=5, use_median=True)
```

To apply the mean windows smoothing, but specifying the window as a time string (in this example, 7 seconds):

```
In [5]: tic3 = window_smooth(tic, window='7s')
```

Write the original TIC and the smoothed TICs to disk:

```
In [6]: tic.write(output_directory / "noise_smoothing_tic.dat", minutes=True)
tic1.write(output_directory / "noise_smoothing_tic1.dat", minutes=True)
tic2.write(output_directory / "noise_smoothing_tic2.dat", minutes=True)
```

Window Averaging on Intensity Matrix

In the previous section, window averaging was applied to an Ion Chromatogram object (in that case a TIC). Where filtering is to be performed on all Ion Chromatograms, the `window_smooth_im()` function may be used instead.

The use of this function is identical to the Ion Chromatogram `window_smooth()` function, except that an Intensity Matrix is passed to it.

For example, to perform window smoothing on an *IntensityMatrix* object with a 5 point window and mean window smoothing:

```
In [7]: from pyms.IntensityMatrix import build_intensity_matrix
from pyms.Noise.Window import window_smooth_im
im = build_intensity_matrix(data)
im_smooth1 = window_smooth_im(im, window=5, use_median=False)
```

Write the IC for mass 73 to disk for both the original and smoothed *IntensityMatrix*:

```
In [8]: ic = im.get_ic_at_index(73)
ic_smooth1 = im_smooth1.get_ic_at_index(73)

ic.write(output_directory/"noise_smoothing_ic.dat", minutes=True)
ic_smooth1.write(output_directory/"noise_smoothing_ic_smooth1.dat", minutes=True)
```

Savitzky-Golay noise filter

A more sophisticated noise filter is the Savitzky-Golay filter. Given the data loaded as above, this filter can be applied as follows:

```
In [9]: from pyms.Noise.SavitzkyGolay import savitzky_golay
tic4 = savitzky_golay(tic)
```

Write the smoothed TIC to disk:

```
In [10]: tic4.write(output_directory / "noise_smoothing_tic4.dat", minutes=True)
```

In this example the default parameters were used.

Savitzky-Golay Noise filtering of Intensity Matrix Object

The `savitzky_golay()` function described above acts on a single `IonChromatogram`. Where it is desired to perform Savitzky Golay filtering on the whole `IntensityMatrix` the function `savitzky_golay_im()` may be used as follows:

```
In [11]: from pyms.Noise.SavitzkyGolay import savitzky_golay_im
         im_smooth2 = savitzky_golay_im(im)
```

Write the IC for mass 73 in the smoothed `IntensityMatrix` to disk:

```
In [12]: ic_smooth2 = im_smooth2.get_ic_at_index(73)
         ic_smooth2.write(output_directory/"noise_smoothing_ic_smooth2.dat", minutes=True)
```

Note: This example is in `pyms-demo/jupyter/NoiseSmoothing.ipynb`.

Baseline Correction

Baseline distortion originating from instrument imperfections and experimental setup is often observed in mass spectrometry data, and off-line baseline correction is often an important step in data pre-processing. There are many approaches for baseline correction. One advanced approach is based on the top-hat transform developed in mathematical morphology¹, and used extensively in digital image processing for tasks such as image enhancement. Top-hat baseline correction was previously applied in proteomics based mass spectrometry². PyMS currently implements only the top-hat baseline corrector, using the SciPy package `ndimage`.

Application of the top-hat baseline corrector requires the size of the structural element to be specified. The structural element needs to be larger than the features one wants to retain in the spectrum after the top-hat transform. In the example below, the top-hat baseline corrector is applied to the TIC of the data set `gc01_0812_066.cdf`, with the structural element of 1.5 minutes:

The purpose of noise smoothing is to remove high-frequency noise from data, and thereby increase the contribution of the signal relative to the contribution of the noise.

First, setup the paths to the datafiles and the output directory, then import `ANDI_reader`, `savitzky_golay` and `tophat`.

```
In [1]: import pathlib
         data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
         # Change this if the data files are stored in a different location

         output_directory = pathlib.Path(".").resolve() / "output"

         from pyms.GCMS.IO.ANDI import ANDI_reader
         from pyms.Noise.SavitzkyGolay import savitzky_golay
         from pyms.TopHat import tophat
```

Read the raw data files and extract the TIC.

```
In [2]: andi_file = data_directory / "gc01_0812_066.cdf"
         data = ANDI_reader(andi_file)
         tic = data.tic
```

¹ Serra J. *Image Analysis and Mathematical Morphology*. Academic Press, Inc, Orlando, 1983. ISBN 0126372403

² Sauve AC and Speed TP. Normalization, baseline correction and alignment of high-throughput mass spectrometry data. *Proceedings Gensips*, 2004

```
-> Reading netCDF file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.cdf'
```

Perform Savitzky-Golay smoothing

```
In [3]: tic1 = savitzky_golay(tic)
```

Perform Tophat baseline correction

```
In [4]: tic2 = tophat(tic1, struct="1.5m")
```

Save the output to disk

```
In [5]: tic.write(output_directory / "baseline_correction_tic.dat", minutes=True)
tic1.write(output_directory / "baseline_correction_tic_smooth.dat", minutes=True)
tic2.write(output_directory / "baseline_correction_tic_smooth_bc.dat", minutes=True)
```

Tophat Baseline correction on an Intensity Matrix object

The `tophat()` function acts on a single *IonChromatogram*. To perform baseline correction on an *IntensityMatrix* object (i.e. on all Ion Chromatograms) the `tophat_im()` function may be used.

Using the same value for `struct` as above, `tophat_im()` is used as follows:

```
In [6]: from pyms.TopHat import tophat_im
from pyms.IntensityMatrix import build_intensity_matrix
im = build_intensity_matrix(data)
im_base_corr = tophat_im(im, struct="1.5m")
```

Write the IC for mass 73 to disk for both the original and smoothed *IntensityMatrix*:

```
In [7]: ic = im.get_ic_at_index(73)
ic_base_corr = im_base_corr.get_ic_at_index(73)

ic.write(output_directory/"baseline_correction_ic.dat", minutes=True)
ic_base_corr.write(output_directory/"baseline_correction_ic_base_corr.dat",
↳ minutes=True)
```

Note: This example is in `pyms-demo/jupyter/BaselineCorrection.ipynb`.

Pre-processing the IntensityMatrix

Noise smoothing and baseline correction can be applied to each *IonChromatogram* in an *IntensityMatrix*.

First, setup the paths to the datafiles and the output directory, then import the required functions.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.GCMS.IO.JCAMP import JCAMP_reader
from pyms.IntensityMatrix import build_intensity_matrix
```

(continues on next page)

(continued from previous page)

```
from pyms.Noise.SavitzkyGolay import savitzky_golay
from pyms.TopHat import tophat
```

Read the raw data files and build the *IntensityMatrix*:

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
im = build_intensity_matrix(data)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

Perform Savitzky-Golay smoothing and Tophat baseline correction

```
In [3]: n_scan, n_mz = im.size

for ii in range(n_mz):
    # print("Working on IC#", ii+1)
    ic = im.get_ic_at_index(ii)
    ic_smooth = savitzky_golay(ic)
    ic_bc = tophat(ic_smooth, struct="1.5m")
    im.set_ic_at_index(ii, ic_bc)
```

Alternatively, the filtering may be performed on the *IntensityMatrix* without using a `for` loop, as outlined in previous examples. However filtering by *IonChromatogram* in a `for` loop as described here is much faster.

The resulting *IntensityMatrix* object can be “dumped” to a file for later retrieval. There are general purpose object file handling methods in *pyms.Utills.IO*. For example;

```
>>> from pyms.Utills.IO import dump_object
>>> dump_object(im, "output/im-proc.dump")
```

Note: This example is in `pyms-demo/jupyter/IntensityMatrix_Preprocessing.ipynb`.

References

6.2.4 Peak detection and representation

Table of Contents

- *Example: Peak Objects*
 - *Creating a Peak Object*
 - *Peak Object properties*
 - *Modifying a Peak Object*
- *Peak Detection*
 - *Example: Peak Detection*
 - *Example: Peak List Filtering*
- *Noise analysis for peak filtering*

- *Peak Area Estimation*
 - *Individual Ion Areas*
 - *Reading the area of a single ion in a peak*
- *References*

Example: Peak Objects

Fundamental to GC-MS analysis is the identification of individual components of the sample mix. The basic component unit is represented as a signal peak. In PyMassSpec a signal peak is represented as *Peak* object. PyMassSpec provides functions to detect peaks and create peaks (discussed at the end of the chapter).

A peak object stores a minimal set of information about a signal peak, namely, the retention time at which the peak apex occurs and the mass spectra at the apex. Additional information, such as, peak width, TIC and individual ion areas can be filtered from the GC-MS data and added to the Peak object information.

Creating a Peak Object

A peak object can be created for a scan at a given retention time by providing the retention time (in minutes or seconds) and the *MassSpectrum* object of the scan.

First, setup the paths to the datafiles and the output directory, then import JCAMP_reader.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pym-s-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pym.s.GCMS.IO.JCAMP import JCAMP_reader
```

Read the raw data files.

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pym-s-data/gc01_0812_066.jdx'
```

Build the *IntensityMatrix*.

```
In [3]: from pym.s.IntensityMatrix import build_intensity_matrix_i

im = build_intensity_matrix_i(data)
```

Extract the *MassSpectrum* at 31.17 minutes in this example.

```
In [4]: index = im.get_index_at_time(31.17*60.0)
ms = im.get_ms_at_index(index)
```

Create a *Peak* object for the given retention time.

```
In [5]: from pym.s.Peak.Class import Peak
peak = Peak(31.17, ms, minutes=True)
```

By default the retention time is assumed to be in seconds. The parameter `minutes` can be set to `True` if the retention time is given in minutes. Internally, PyMassSpec stores retention times in seconds, so the `minutes` parameter ensures the input and output of the retention time are in the same units.

Peak Object properties

The retention time of the peak, in seconds, can be returned with `pym.s.Peak.Class.Peak.rt`. The mass spectrum can be returned with `pym.s.Peak.Class.Peak.mass_spectrum`.

The `Peak` object constructs a unique identification (UID) based on the spectrum and retention time. This helps in managing lists of peaks (covered in the next chapter). The UID can be returned with `pym.s.Peak.Class.Peak.UID`. The format of the UID is the masses of the two most abundant ions in the spectrum, the ratio of the abundances of the two ions, and the retention time (in the same units as given when the Peak object was created). The format is:

Mass1-Mass2-Ratio-RT

For example:

```
In [6]: peak.rt
```

```
1870.2
```

```
In [7]: peak.UID
```

```
'319-73-74-1870.20'
```

```
In [8]: index = im.get_index_of_mass(73.3)
```

```
index
```

```
23
```

Modifying a Peak Object

The `Peak` object has methods for modifying the mass spectrum. The mass range can be cropped to a smaller range with `crop_mass()`, and the intensity values for a single ion can be set to zero with `null_mass()`. For example, the mass range can be set from 60 to 450 m/z , and the ions related to sample preparation can be ignored by setting their intensities to zero as follows:

```
In [9]: peak.crop_mass(60, 450)
        peak.null_mass(73)
        peak.null_mass(147)
```

The UID is automatically updated to reflect the changes:

```
In [10]: peak.UID
```

```
'319-205-54-1870.20'
```

It is also possible to change the peak mass spectrum by setting the attribute `pym.s.Peak.Class.Peak.mass_spectrum`.

Note: This example is in `pyms-demo/jupyter/Peak.ipynb`.

Peak Detection

The general use of a *Peak* object is to extract them from the GC-MS data and build a list of peaks. In PyMassSpec, the function for peak detection is based on the method of Biller and Biemann (1974)¹. The basic process is to find all maximising ions in a pre-set window of scans, for a given scan. The ions that maximise at a given scan are taken to belong to the same peak.

The function is *BillerBiemann()*. in `pyms.BillerBiemann`. The function has parameters for the window width for detecting the local maxima (`points`), and the number of scans across which neighbouring, apexing, ions are combined and considered as belonging to the same peak. The number of neighbouring scans to combine is related to the likelihood of detecting a peak apex at a single scan or several neighbouring scans. This is more likely when there are many scans across the peak. It is also possible, however, when there are very few scans across the peak. The scans are combined by taking all apexing ions to have occurred at the scan that had to greatest TIC prior to combining scans.

Example: Peak Detection

First, setup the paths to the datafiles and the output directory, then import JCAMP_reader and build_intensity_matrix.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.GCMS.IO.JCAMP import JCAMP_reader
from pyms.IntensityMatrix import build_intensity_matrix
```

Read the raw data file and build the *IntensityMatrix*.

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
im = build_intensity_matrix(data)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

Preprocess the data (Savitzky-Golay smoothing and Tophat baseline detection).

```
In [3]: from pyms.Noise.SavitzkyGolay import savitzky_golay
from pyms.TopHat import tophat

n_scan, n_mz = im.size

for ii in range(n_mz):
    ic = im.get_ic_at_index(ii)
    ic_smooth = savitzky_golay(ic)
    ic_bc = tophat(ic_smooth, struct="1.5m")
    im.set_ic_at_index(ii, ic_bc)
```

¹ Biller JE and Biemann K. Reconstructed mass spectra, a novel approach for the utilization of gas chromatograph-mass spectrometer data. *Anal. Lett.*, 7:515-528, 1974

Now the Biller and Biemann based technique can be applied to detect peaks.

```
In [4]: from pyms.BillerBiemann import BillerBiemann
peak_list = BillerBiemann(im)
peak_list[:10]
```

```
[<pyms.Peak.Class.Peak at 0x7f8deca882b0>,
 <pyms.Peak.Class.Peak at 0x7f8deca88550>,
 <pyms.Peak.Class.Peak at 0x7f8dce908160>,
 <pyms.Peak.Class.Peak at 0x7f8df3bc5c50>,
 <pyms.Peak.Class.Peak at 0x7f8dc3a043c8>,
 <pyms.Peak.Class.Peak at 0x7f8dc3a04588>,
 <pyms.Peak.Class.Peak at 0x7f8dc3a045f8>,
 <pyms.Peak.Class.Peak at 0x7f8dc3a04668>,
 <pyms.Peak.Class.Peak at 0x7f8dc3a046d8>,
 <pyms.Peak.Class.Peak at 0x7f8dc3a04748>]
```

```
In [5]: len(peak_list)
```

```
9845
```

Note that this is nearly as many peaks as there are scans in the data (9865 scans). This is due to noise and the simplicity of the technique.

The number of detected peaks can be constrained by the selection of better parameters. Parameters can be determined by counting the number of points across a peak, and examining where peaks are found. For example, the peak list can be found with the parameters of a window of 9 points and by combining 2 neighbouring scans if they apex next to each other:

```
In [6]: peak_list = BillerBiemann(im, points=9, scans=2)
peak_list[:10]
```

```
[<pyms.Peak.Class.Peak at 0x7f8dae545be0>,
 <pyms.Peak.Class.Peak at 0x7f8dae545c18>,
 <pyms.Peak.Class.Peak at 0x7f8dae545c88>,
 <pyms.Peak.Class.Peak at 0x7f8dae545cf8>,
 <pyms.Peak.Class.Peak at 0x7f8dae545d68>,
 <pyms.Peak.Class.Peak at 0x7f8dae545dd8>,
 <pyms.Peak.Class.Peak at 0x7f8dae545e48>,
 <pyms.Peak.Class.Peak at 0x7f8dae545eb8>,
 <pyms.Peak.Class.Peak at 0x7f8dae545f28>,
 <pyms.Peak.Class.Peak at 0x7f8dae545f98>]
```

```
In [7]: len(peak_list)
```

```
3695
```

The number of detected peaks has been reduced, but there are still many more than would be expected from the sample. Functions to filter the peak list are covered in the next example.

Example: Peak List Filtering

There are two functions to filter the list of Peak objects.

The first, `rel_threshold()` modifies the mass spectrum stored in each peak so any intensity that is less than a given percentage of the maximum intensity for the peak is removed.

The second, `num_ions_threshold()`, removes any peak that has less than a given number of ions above a given threshold.

Once the peak list has been constructed, the filters can be applied as follows:

```
In [8]: from pyms.BillerBiemann import rel_threshold, num_ions_threshold
pl = rel_threshold(peak_list, percent=2)
pl[:10]
```

```
[<pyms.Peak.Class.Peak at 0x7f8dc3a045f8>,
<pyms.Peak.Class.Peak at 0x7f8dc3a04630>,
<pyms.Peak.Class.Peak at 0x7f8dc3a04748>,
<pyms.Peak.Class.Peak at 0x7f8dc3a047b8>,
<pyms.Peak.Class.Peak at 0x7f8dc3a048d0>,
<pyms.Peak.Class.Peak at 0x7f8dc3a049e8>,
<pyms.Peak.Class.Peak at 0x7f8dc3a04a20>,
<pyms.Peak.Class.Peak at 0x7f8dc3a04b38>,
<pyms.Peak.Class.Peak at 0x7f8dc3a04b70>,
<pyms.Peak.Class.Peak at 0x7f8dc3a04c88>]
```

```
In [9]: new_peak_list = num_ions_threshold(pl, n=3, cutoff=10000)
new_peak_list[:10]
```

```
[<pyms.Peak.Class.Peak at 0x7f8deca8e128>,
<pyms.Peak.Class.Peak at 0x7f8deca8e1d0>,
<pyms.Peak.Class.Peak at 0x7f8dc3a04780>,
<pyms.Peak.Class.Peak at 0x7f8dc3a04550>,
<pyms.Peak.Class.Peak at 0x7f8dbb3cf3c8>,
<pyms.Peak.Class.Peak at 0x7f8dbb3cf048>,
<pyms.Peak.Class.Peak at 0x7f8dbb3cf4a8>,
<pyms.Peak.Class.Peak at 0x7f8dbb3cf550>,
<pyms.Peak.Class.Peak at 0x7f8dbb3cf5f8>,
<pyms.Peak.Class.Peak at 0x7f8dbb3cf6a0>]
```

```
In [10]: len(new_peak_list)
```

```
146
```

The number of detected peaks is now more realistic of what would be expected in the test sample.

Note: This example is in `pyms-demo/jupyter/Peak_Detection.ipynb`.

Noise analysis for peak filtering

In the previous example the cutoff parameter for peak filtering was set by the user. This can work well for individual data files, but can cause problems when applied to large experiments with many individual data files. Where experimental conditions have changed slightly between experimental runs, the ion intensity over the GC-MS run may also change. This means that an inflexible cutoff value can work for some data files, while excluding too many, or including too many peaks in other files.

An alternative to manually setting the value for cutoff is to use the `window_analyzer()` function. This function examines a Total Ion Chromatogram (TIC) and computes a value for the median absolute deviation in troughs between peaks. This gives an approximate threshold value above which false peaks from noise should be filtered out.

First, build the Peak list as before

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pym-s-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pym.s.GCMS.IO.JCAMP import JCAMP_reader
from pym.s.IntensityMatrix import build_intensity_matrix
from pym.s.Noise.SavitzkyGolay import savitzky_golay
from pym.s.TopHat import tophat
from pym.s.BillerBiemann import BillerBiemann

jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
im = build_intensity_matrix(data)

n_scan, n_mz = im.size

for ii in range(n_mz):
    ic = im.get_ic_at_index(ii)
    ic_smooth = savitzky_golay(ic)
    ic_bc = tophat(ic_smooth, struct="1.5m")
    im.set_ic_at_index(ii, ic_bc)

peak_list = BillerBiemann(im, points=9, scans=2)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pym-s-data/gc01_0812_066.jdx'
```

Compute the noise value.

```
In [2]: from pym.s.Noise.Analysis import window_analyzer

tic = data.tic

noise_level = window_analyzer(tic)
noise_level
```

```
432.1719792438844
```

Filter the Peak List using this noise value as the cutoff.

```
In [3]: from pym.s.BillerBiemann import num_ions_threshold
filtered_peak_list = num_ions_threshold(peak_list, n=3, cutoff=noise_level)
filtered_peak_list[:10]
```

```
[<pyms.Peak.Class.Peak at 0x7f4a9864f128>,  
<pyms.Peak.Class.Peak at 0x7f4a9864f2e8>,  
<pyms.Peak.Class.Peak at 0x7f4a9864f320>,  
<pyms.Peak.Class.Peak at 0x7f4a9864f3c8>,  
<pyms.Peak.Class.Peak at 0x7f4a9864f518>,  
<pyms.Peak.Class.Peak at 0x7f4a9864f4a8>,  
<pyms.Peak.Class.Peak at 0x7f4a9864f6a0>,  
<pyms.Peak.Class.Peak at 0x7f4a9864f748>,  
<pyms.Peak.Class.Peak at 0x7f4a9864f7f0>,  
<pyms.Peak.Class.Peak at 0x7f4a9864f898>]
```

```
In [4]: len(filtered_peak_list)
```

```
612
```

Note: This example is in `pyms-demo/jupyter/Peak_Filtering_Noise_Analysis.ipynb`.

Peak Area Estimation

The *Peak* object does not contain any information about the width or area of the peak when it is first created. This information can be added after the instantiation of a Peak object. The area of the peak can be set with the attribute *area*.

The total peak area can be obtained by the `peak_sum_area()` function in `pyms.Peak.Function`. The function determines the total area as the sum of the ion intensities for all masses that apex at the given peak. To calculate the peak area of a single mass, the intensities are added from the apex of the mass peak outwards.

Edge values are added until the following conditions are met:

- the added intensity adds less than 0.5% to the accumulated area; or
- the added intensity starts increasing (i.e. when the ion is common to co-eluting compounds).

To avoid noise effects, the edge value is taken at the midpoint of three consecutive edge values.

First, build the Peak list as before

```
In [1]: import pathlib  
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"  
# Change this if the data files are stored in a different location  
  
output_directory = pathlib.Path(".").resolve() / "output"  
  
from pyms.GCMS.IO.JCAMP import JCAMP_reader  
from pyms.IntensityMatrix import build_intensity_matrix  
from pyms.Noise.SavitzkyGolay import savitzky_golay  
from pyms.TopHat import tophat  
from pyms.BillerBiemann import BillerBiemann  
  
jcamp_file = data_directory / "gc01_0812_066.jdx"  
data = JCAMP_reader(jcamp_file)  
im = build_intensity_matrix(data)  
  
n_scan, n_mz = im.size
```

(continues on next page)

(continued from previous page)

```

for ii in range(n_mz):
    ic = im.get_ic_at_index(ii)
    ic_smooth = savitzky_golay(ic)
    ic_bc = tophat(ic_smooth, struct="1.5m")
    im.set_ic_at_index(ii, ic_bc)

peak_list = BillerBiemann(im, points=9, scans=2)

from pyms.Noise.Analysis import window_analyzer
tic = data.tic
noise_level = window_analyzer(tic)

from pyms.BillerBiemann import num_ions_threshold
filtered_peak_list = num_ions_threshold(peak_list, n=3, cutoff=noise_level)
filtered_peak_list[:10]

```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

```

[<pyms.Peak.Class.Peak at 0x7fa8eae80198>,
 <pyms.Peak.Class.Peak at 0x7fa8eae80208>,
 <pyms.Peak.Class.Peak at 0x7fa8eae802b0>,
 <pyms.Peak.Class.Peak at 0x7fa8eae80358>,
 <pyms.Peak.Class.Peak at 0x7fa8eae80400>,
 <pyms.Peak.Class.Peak at 0x7fa8eae804a8>,
 <pyms.Peak.Class.Peak at 0x7fa8eae80550>,
 <pyms.Peak.Class.Peak at 0x7fa8eae805f8>,
 <pyms.Peak.Class.Peak at 0x7fa8eae806a0>,
 <pyms.Peak.Class.Peak at 0x7fa8eae80748>]

```

Given a list of peaks, areas can be determined and added as follows:

```

In [2]: from pyms.Peak.Function import peak_sum_area
        for peak in peak_list:
            area = peak_sum_area(im, peak)
            peak.area = area

```

Note: This example is in `pyms-demo/jupyter/Peak_Area_Estimation.ipynb`.

Individual Ion Areas

Note: This example is in `pyms-demo/56`

While the previous approach uses the sum of all areas in the peak to estimate the peak area, the user may also choose to record the area of each individual ion in each peak.

This can be useful when the intention is to later perform quantitation based on the area of a single characteristic ion for a particular compound. It is also essential if using the Common Ion Algorithm for quantitation, outlined in the section `common-ion`.

To set the area of each ion for each peak, the following code is used:

```
>>> from pyms.Peak.Function import peak_top_ion_areas
>>> for peak in peak_list:
...     area_dict = peak_top_ions_areas(intensity_matrix, peak)
...     peak.set_ion_areas(area_dict)
...
```

This will set the areas of the 5 most abundant ions in each peak. If it is desired to record more than the top five ions, the argument `num_ions=x` should be supplied, where `x` is the number of most abundant ions to be recorded. For example:

```
...     area_dict = peak_top_ions_areas(intensity_matrix, peak, num_ions=10)
```

will record the 10 most abundant ions for each peak.

The individual ion areas can be set instead of, or in addition to the total area for each peak.

Reading the area of a single ion in a peak

If the individual ion areas have been set for a peak, it is possible to read the area of an individual ion for the peak. For example:

```
>>> peak.get_ion_area(101)
```

will return the area of the m/z value 101 for the peak. If the area of that ion has not been set (i.e. it was not one of the most abundant ions), the function will return `None`.

References

6.2.5 Peak alignment by dynamic programming

Table of Contents

- *Preparation of multiple experiments for peak alignment by dynamic programming*
 - *Example: Creating an Experiment*
 - *Example: Creating Multiple Experiments*
- *Dynamic Programming Alignment*
 - *Example: Within-state alignment of peak lists from multiple experiments*
 - *Example: Between-state alignment of peak lists from multiple experiments*
- *Common Ion Area Quantitation*
 - *Using the Common Ion Algorithm*
- *References*

PyMS provides functions to align GC-MS peaks by dynamic programming¹. The peak alignment by dynamic programming uses both peak apex retention time and mass spectra. This information is determined from the raw GC-MS

¹ Robinson MD, De Souza DP, Keen WW, Saunders EC, McConville MJ, Speed TP, and Likic VA. A dynamic programming approach for the alignment of signal peaks in multiple gas chromatography-mass spectrometry experiments. *BMC Bioinformatics*, 8:419, 2007

data by applying a series of processing steps to produce data that can then be aligned and used for statistical analysis. The details are described in this chapter.

Preparation of multiple experiments for peak alignment by dynamic programming

Example: Creating an Experiment

Before aligning peaks from multiple experiments, the peak objects need to be created and encapsulated into *Experiment* objects. During this process it is often useful to pre-process the peaks in some way, for example to null certain m/z channels and/or to select a certain retention time range.

The procedure starts the same as in the previous examples, namely:

1. read a file,
2. bin the data into fixed mass values,
3. smooth the data,
4. remove the baseline,
5. deconvolute peaks,
6. filter the peaks,
7. set the mass range,
8. remove uninformative ions, and
9. estimate peak areas.

First, setup the paths to the datafiles and the output directory, then import *ANDI_reader* and *build_intensity_matrix_i*.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pym-s-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pym.s.GCMS.IO.ANDI import ANDI_reader
from pym.s.IntensityMatrix import build_intensity_matrix_i
```

Read the raw data file and build the *IntensityMatrix*.

```
In [2]: andi_file = data_directory / "a0806_077.cdf"
data = ANDI_reader(andi_file)
im = build_intensity_matrix_i(data)
```

```
-> Reading netCDF file '/home/vagrant/PyMassSpec/pym-s-data/a0806_077.cdf'
```

Preprocess the data (Savitzky-Golay smoothing and Tophat baseline detection)

```
In [3]: from pym.s.Noise.SavitzkyGolay import savitzky_golay
from pym.s.TopHat import tophat

n_scan, n_mz = im.size

for ii in range(n_mz):
    ic = im.get_ic_at_index(ii)
```

(continues on next page)

(continued from previous page)

```
ic1 = savitzky_golay(ic)
ic_smooth = savitzky_golay(ic1) # Why the second pass here?
ic_bc = tophat(ic_smooth, struct="1.5m")
im.set_ic_at_index(ii, ic_bc)
```

Now the Biller and Biemann based technique can be applied to detect peaks.

```
In [4]: from pyms.BillerBiemann import BillerBiemann
pl = BillerBiemann(im, points=9, scans=2)
len(pl)
```

```
1191
```

Trim the peak list by relative intensity

```
In [5]: from pyms.BillerBiemann import rel_threshold, num_ions_threshold
apl = rel_threshold(pl, percent=2)
len(apl)
```

```
1191
```

Trim the peak list by noise threshold

```
In [6]: peak_list = num_ions_threshold(apl, n=3, cutoff=3000)
len(peak_list)
```

```
225
```

Set the mass range, remove unwanted ions and estimate the peak area

```
In [7]: from pyms.Peak.Function import peak_sum_area

for peak in peak_list:
    peak.crop_mass(51, 540)

    peak.null_mass(73)
    peak.null_mass(147)

    area = peak_sum_area(im, peak)
    peak.area = area
```

Create an *Experiment*.

```
In [8]: from pyms.Experiment import Experiment

expr = Experiment("a0806_077", peak_list)
```

Set the time range for all Experiments

```
In [9]: expr.sele_rt_range(["6.5m", "21m"])
```

Save the experiment to disk.

```
In [10]: expr.dump(output_directory / "experiments" / "a0806_077.expr")
```

Note: This example is in `pyms-demo/jupyter/Experiment.ipynb`.

Example: Creating Multiple Experiments

In example three GC-MS experiments are prepared for peak alignment. The experiments are named `a0806_077`, `a0806_078`, `a0806_079`, and represent separate GC-MS sample runs from the same biological sample.

The procedure is the same as for the previous example, but is repeated three times.

First, setup the paths to the datafiles and the output directory, then import the required functions.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.BillerBiemann import BillerBiemann, num_ions_threshold, rel_threshold
from pyms.Experiment import Experiment
from pyms.GCMS.IO.ANDI import ANDI_reader
from pyms.IntensityMatrix import build_intensity_matrix_i
from pyms.Noise.SavitzkyGolay import savitzky_golay
from pyms.Peak.Function import peak_sum_area, peak_top_ion_areas
from pyms.TopHat import tophat
```

Define the data files to process

```
In [2]: expr_codes = ["a0806_077", "a0806_078", "a0806_079"]
# expr_codes = ["a0806_140", "a0806_141", "a0806_142"]
```

Loop over the experiments and perform the processing.

```
In [3]: for expr_code in expr_codes:

    print(f" -> Processing experiment '{expr_code}'")

    andi_file = data_directory / f"{expr_code}.cdf"

    data = ANDI_reader(andi_file)

    im = build_intensity_matrix_i(data)

    n_scan, n_mz = im.size

    # Preprocess the data (Savitzky-Golay smoothing and Tophat baseline detection)

    for ii in range(n_mz):
        ic = im.get_ic_at_index(ii)
        ic1 = savitzky_golay(ic)
        ic_smooth = savitzky_golay(ic1) # Why the second pass here?
        ic_bc = tophat(ic_smooth, struct="1.5m")
        im.set_ic_at_index(ii, ic_bc)

    # Peak detection
    pl = BillerBiemann(im, points=9, scans=2)
```

(continues on next page)

(continued from previous page)

```

# Trim the peak list by relative intensity
apl = rel_threshold(pl, percent=2)

# Trim the peak list by noise threshold
peak_list = num_ions_threshold(apl, n=3, cutoff=3000)

print("\t -> Number of Peaks found:", len(peak_list))

print("\t -> Executing peak post-processing and quantification...")

# Set the mass range, remove unwanted ions and estimate the peak area
# For peak alignment, all experiments must have the same mass range

for peak in peak_list:
    peak.crop_mass(51, 540)

    peak.null_mass(73)
    peak.null_mass(147)

    area = peak_sum_area(im, peak)
    peak.area = area
    area_dict = peak_top_ion_areas(im, peak)
    peak.ion_areas = area_dict

# Create an Experiment
expr = Experiment(expr_code, peak_list)

# Use the same retention time range for all experiments
lo_rt_limit = "6.5m"
hi_rt_limit = "21m"

print(f"\t -> Selecting retention time range between '{lo_rt_limit}' and '{hi_rt_
->limit}'")

expr.sele_rt_range([lo_rt_limit, hi_rt_limit])

# Save the experiment to disk.
output_file = output_directory / "experiments" / f"{expr_code}.expr"
print(f"\t -> Saving the result as '{output_file}'")

expr.dump(output_file)

```

```

-> Processing experiment 'a0806_077'
-> Reading netCDF file '/home/vagrant/PyMassSpec/pyms-data/a0806_077.cdf'
    -> Number of Peaks found: 225
    -> Executing peak post-processing and quantification...
    -> Selecting retention time range between '6.5m' and '21m'
    -> Saving the result as '/home/vagrant/PyMassSpec/pyms-demo/jupyter/output/
->experiments/a0806_077.expr'
-> Processing experiment 'a0806_078'
-> Reading netCDF file '/home/vagrant/PyMassSpec/pyms-data/a0806_078.cdf'
    -> Number of Peaks found: 238
    -> Executing peak post-processing and quantification...
    -> Selecting retention time range between '6.5m' and '21m'
    -> Saving the result as '/home/vagrant/PyMassSpec/pyms-demo/jupyter/output/
->experiments/a0806_078.expr'

```

(continues on next page)

(continued from previous page)

```
-> Processing experiment 'a0806_079'
-> Reading netCDF file '/home/vagrant/PyMassSpec/pyms-data/a0806_079.cdf'
    -> Number of Peaks found: 268
    -> Executing peak post-processing and quantification...
    -> Selecting retention time range between '6.5m' and '21m'
    -> Saving the result as '/home/vagrant/PyMassSpec/pyms-demo/jupyter/output/
->experiments/a0806_079.expr'
```

The previous set of data all belong to the same experimental condition. That is, they represent one group and any comparison between the data is a within group comparison. For the original experiment, another set of GC-MS data was collected for a different experimental condition. This group must also be stored as a set of experiments, and can be used for between group comparison.

The second set of data files are named a0806_140, a0806_141, and a0806_142, and are processed and stored as above.

In the example notebook, you can uncomment the line in code cell 2 and run the notebook again to process the second set of data files.

Note: This example is in `pyms-demo/jupyter/Multiple_Experiments.ipynb`.

Dynamic Programming Alignment

Example: Within-state alignment of peak lists from multiple experiments

In this example the experiments a0806_077, a0806_078, and a0806_079 prepared in the previous example will be aligned, and therefore the notebook `Multiple_Experiments.ipynb` must be run first to create the files a0806_077.expr, a0806_078.expr, a0806_079.expr. These files contain the post-processed peak lists from the three experiments.

First, determine the directory to the experiment files and import the required functions.

```
In [1]: import pathlib
output_directory = pathlib.Path(".").resolve() / "output"

from pyms.DPA.PairwiseAlignment import PairwiseAlignment, align_with_tree
from pyms.DPA.Alignment import expr12alignment
from pyms.Experiment import load_expr
```

Define the input experiments list.

```
In [2]: exprA_codes = ["a0806_077", "a0806_078", "a0806_079"]
```

Read the experiment files from disk and create a list of the loaded *Experiment* objects.

```
In [3]: expr_list = []

for expr_code in exprA_codes:
    file_name = output_directory / "experiments" / f"{expr_code}.expr"
    expr = load_expr(file_name)
    expr_list.append(expr)
```

Define the within-state alignment parameters.

```
In [4]: Dw = 2.5 # rt modulation [s]
        Gw = 0.30 # gap penalty
```

Convert each *Experiment* object is converted into an *Alignment* object with the function `exprl2alignment()`.

```
In [5]: F1 = exprl2alignment(expr_list)
```

In this example, there is only one experimental condition so the alignment object is only for within group alignment (this special case is called 1-alignment). The variable `F1` is a Python list containing three alignment objects.

Perform pairwise alignment. The class `lpys.DPA.Class.PairwiseAlignment` calculates the similarity between all peaks in one sample with those of another sample. This is done for all possible pairwise alignments (2-alignments).

```
In [6]: T1 = PairwiseAlignment(F1, Dw, Gw)
```

```
Calculating pairwise alignments for 3 alignments (D=2.50, gap=0.30)
-> 2 pairs remaining
-> 1 pairs remaining
-> 0 pairs remaining
-> Clustering 6 pairwise alignments.Done
```

The parameters for the alignment by dynamic programming are: `Dw`, the retention time modulation in seconds; and `Gw`, the gap penalty. These parameters are explained in detail in [1].

The output of `PairwiseAlignment` (`T1`) is an object which contains the dendrogram tree that maps the similarity relationship between the input 1-alignments, and also 1-alignments themselves.

The function `align_with_tree()` then takes the object `T1` and aligns the individual alignment objects according to the guide tree.

```
In [7]: A1 = align_with_tree(T1, min_peaks=2)
```

```
Aligning 3 items with guide tree (D=2.50, gap=0.30)
-> 1 item(s) remaining
-> 0 item(s) remaining
```

In this example, the individual alignments are three 1-alignments, and the function `align_with_tree()` first creates a 2-alignment from the two most similar 1-alignments and then adds the third 1-alignment to this to create a 3-alignment.

The parameter `min_peaks=2` specifies that any peak column of the data matrix that has fewer than two peaks in the final alignment will be dropped. This is useful to clean up the data matrix of accidental peaks that are not truly observed over the set of replicates.

Finally, the resulting 3-alignment is saved by writing alignment tables containing peak retention times (`rt.csv`) and the corresponding peak areas (`area.csv`). These are plain ASCII files in CSV format.

```
In [8]: A1.write_csv(
        output_directory / "within_state_alignment" / 'a_rt.csv',
        output_directory / "within_state_alignment" / 'a_area.csv',
        )
```

The file `area1.csv` contains the data matrix where the corresponding peaks are aligned in the columns and each row corresponds to an experiment. The file `rt1.csv` is useful for manually inspecting the alignment.

Example: Between-state alignment of peak lists from multiple experiments

In the previous example the list of peaks were aligned within a single experiment with multiple replicates (“within-state alignment”). In practice, it is of more interest to compare the two experimental states.

In a typical experimental setup there can be multiple replicate experiments on each experimental state or condition. To analyze the results of such an experiment statistically, the list of peaks need to be aligned within each experimental state and also between the states. The result of such an alignment would be the data matrix of integrated peak areas. The data matrix contains a row for each sample and the number of columns is determined by the number of unique peaks (metabolites) detected in all the experiments.

In principle, all experiments could be aligned across conditions and replicates in the one process. However, a more robust approach is to first align experiments within each set of replicates (within-state alignment), and then to align the resulting alignments (between-state alignment) [1].

This example demonstrates how the peak lists from two cell states are aligned.

- Cell state A, consisting of three aligned experiments (a0806_077, a0806_078, and a0806_079), and
- Cell state B, consisting of three aligned experiments (a0806_140, a0806_141, and a0806_142).

These experiments were created in the notebook `Multiple_Experiments.ipynb`.

First, perform within-state alignment for cell state B.

```
In [9]: exprB_codes = ["a0806_140", "a0806_141", "a0806_142"]

expr_list = []

for expr_code in exprB_codes:
    file_name = output_directory / "experiments" / f"{expr_code}.expr"
    expr = load_expr(file_name)
    expr_list.append(expr)

F2 = exprl2alignment(expr_list)
T2 = PairwiseAlignment(F2, Dw, Gw)
A2 = align_with_tree(T2, min_peaks=2)

A2.write_csv(
    output_directory / "within_state_alignment" / 'b_rt.csv',
    output_directory / "within_state_alignment" / 'b_area.csv',
)
```

```
Calculating pairwise alignments for 3 alignments (D=2.50, gap=0.30)
-> 2 pairs remaining
-> 1 pairs remaining
-> 0 pairs remaining
-> Clustering 6 pairwise alignments.Done
Aligning 3 items with guide tree (D=2.50, gap=0.30)
-> 1 item(s) remaining
-> 0 item(s) remaining
```

A1 and A2 are the results of the within group alignments for cell state A and B, respectively. The between-state alignment can be performed as follows alignment commands:

```
In [10]: # Define the within-state alignment parameters.
Db = 10.0 # rt modulation
Gb = 0.30 # gap penalty
```

(continues on next page)

(continued from previous page)

```
T9 = PairwiseAlignment([A1,A2], Db, Gb)
A9 = align_with_tree(T9)

A9.write_csv(
    output_directory / "between_state_alignment" / 'rt.csv',
    output_directory / "between_state_alignment" / 'area.csv')
```

```
Calculating pairwise alignments for 2 alignments (D=10.00, gap=0.30)
-> 0 pairs remaining
-> Clustering 2 pairwise alignments.Done
Aligning 2 items with guide tree (D=10.00, gap=0.30)
-> 0 item(s) remaining
```

Store the aligned peaks to disk.

```
In [11]: from pyms.Peak.List.IO import store_peaks

aligned_peaks = A9.aligned_peaks()
store_peaks(aligned_peaks, output_directory / "between_state_alignment" / 'peaks.bin')
```

In this example the retention time tolerance for between-state alignment is greater compared to the retention time tolerance for the within-state alignment as we expect less fidelity in retention times between them. The same functions are used for the within-state and between-state alignment. The result of the alignment is saved to a file as the area and retention time matrices (described above).

Note: These examples are in `pyms-demo/jupyter/DPA.ipynb`.

Common Ion Area Quantitation

Note: This example is in *pyms-demo/64*

The `area.csv` file produced in the preceding section lists the total area of each peak in the alignment. The total area is the sum of the areas of each of the individual ions in the peak. While this approach produces broadly accurate results, it can result in errors where neighbouring peaks or unfiltered noise add to the peak in some way.

One alternative to this approach is to pick a single ion which is common to a particular peak (compound), and to report only the area of this ion for each occurrence of that peak in the alignment. Using the method `common_ion()` of the class `Alignment`, PyMassSpec can select an ion for each aligned peak which is both abundant and occurs most often for that peak. We call this the ‘Common Ion Algorithm’ (CIA).

To use this method it is essential that the individual ion areas have been set (see section `individual_ion_areas`).

Using the Common Ion Algorithm

When using the CIA for area quantitation, a different method of the class *Alignment* is used to write the area matrix; *write_common_ion_csv()*. This requires a list of the common ions for each peak in the alignment. This list is generated using the Alignment class method *common_ion()*.

Continuing from the previous example, the following invokes common ion filtering on previously created alignment object 'A9':

```
>>> common_ion_list = A9.common_ion()
```

The variable 'common_ion_list' is a list of the common ion for each peak in the alignment. This list is the same length as the alignment. To write peak areas using common ion quantitation:

```
>>> A9.write_common_ion_csv('output/area_common_ion.csv', common_ion_list)
```

References

6.2.6 The Display Module

Table of Contents

- *Example: Displaying a TIC*
- *Example: Displaying Multiple IonChromatogram Objects*
- *Example: Displaying a Mass Spectrum*
- *Example: Displaying Detected Peaks*
- *Example: User Interaction With The Plot Window*

PyMassSpec has graphical capabilities to display information such as *IonChromatogram* objects (ICs), Total Ion Chromatograms (TICs), and detected lists of Peaks.

Example: Displaying a TIC

First, setup the paths to the datafiles and the output directory, then import JCAMP_reader.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pym-s-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pym.GCMS.IO.JCAMP import JCAMP_reader
```

Read the raw data files and extract the TIC

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
tic = data.tic
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

Import matplotlib and the `plot_ic()` function, create a subplot, and plot the TIC:

```
In [3]: import matplotlib.pyplot as plt
        from pyms.Display import plot_ic

        %matplotlib inline
        # Change to ``notebook`` for an interactive view

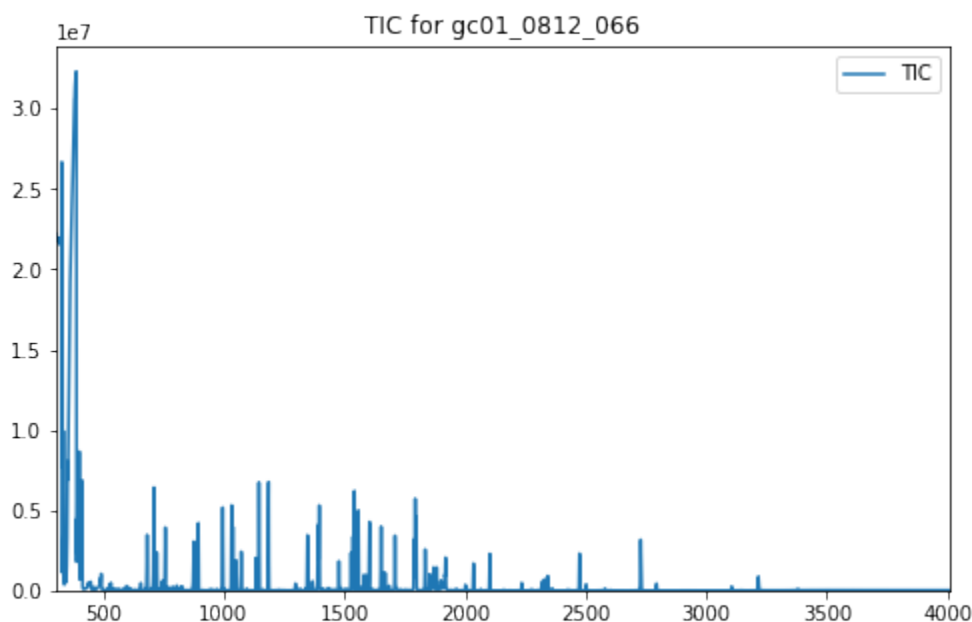
        fig, ax = plt.subplots(1, 1, figsize=(8, 5))

        # Plot the TIC
        plot_ic(ax, tic, label="TIC")

        # Set the title
        ax.set_title("TIC for gc01_0812_066")

        # Add the legend
        plt.legend()

        plt.show()
```



In addition to the TIC, other arguments may be passed to `plot_ic()`. These can adjust the line colour or the text of the legend entry. See https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.lines.Line2D.html for a full list of the possible arguments.

An *IonChromatogram* can be plotted in the same manner as the TIC in the example above.

When not running in Jupyter Notebook, the plot may appear in a separate window looking like this:

Note: This example is in `pyms-demo/jupyter/Displaying_TIC.ipynb` and `pyms-demo/70a/proc.py`.

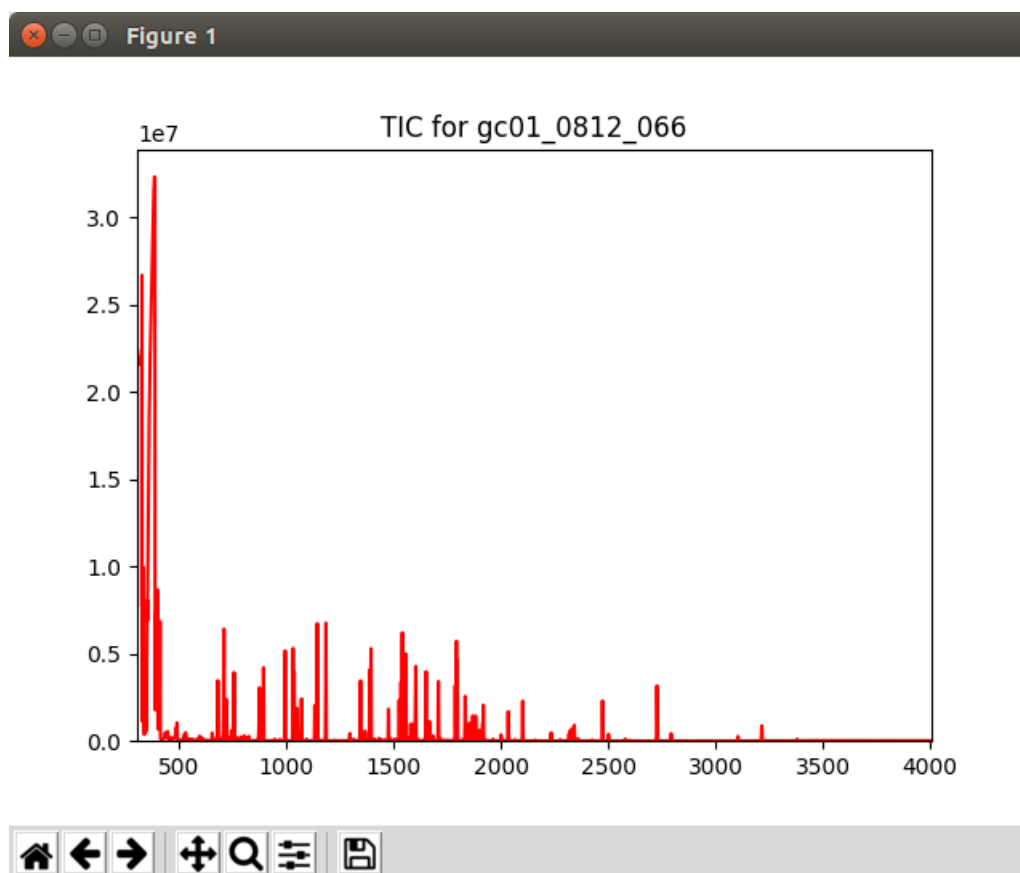


Fig. 1: Graphics window displayed by the script 70a/proc.py

Example: Displaying Multiple IonChromatogram Objects

Multiple *IonChromatogram* objects can be plotted on the same figure.

To start, load a datafile and create an *IntensityMatrix* as before.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.GCMS.IO.JCAMP import JCAMP_reader
from pyms.IntensityMatrix import build_intensity_matrix_i

jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
tic = data.tic
im = build_intensity_matrix_i(data)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

Extract the desired IonChromatograms from the *IntensityMatrix*.

```
In [2]: ic73 = im.get_ic_at_mass(73)
ic147 = im.get_ic_at_mass(147)
```

Import matplotlib and the *plot_ic()* function, create a subplot, and plot the ICs on the chart:

```
In [3]: import matplotlib.pyplot as plt
from pyms.Display import plot_ic

%matplotlib inline
# Change to ``notebook`` for an interactive view

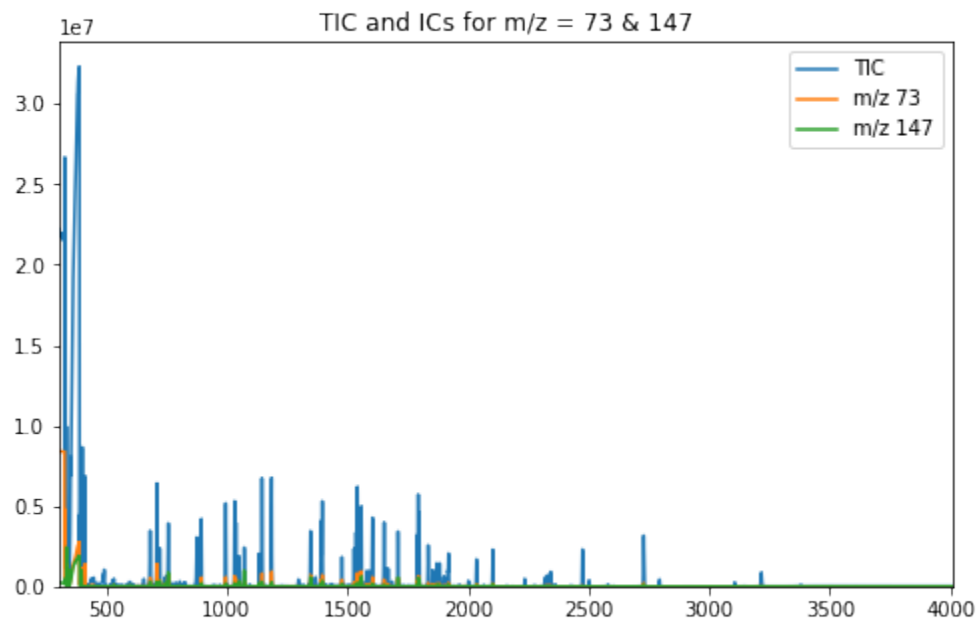
fig, ax = plt.subplots(1, 1, figsize=(8, 5))

# Plot the ICs
plot_ic(ax, tic, label="TIC")
plot_ic(ax, ic73, label="m/z 73")
plot_ic(ax, ic147, label="m/z 147")

# Set the title
ax.set_title("TIC and ICs for m/z = 73 & 147")

# Add the legend
plt.legend()

plt.show()
```



When not running in Jupyter Notebook, the plot may appear in a separate window looking like this:

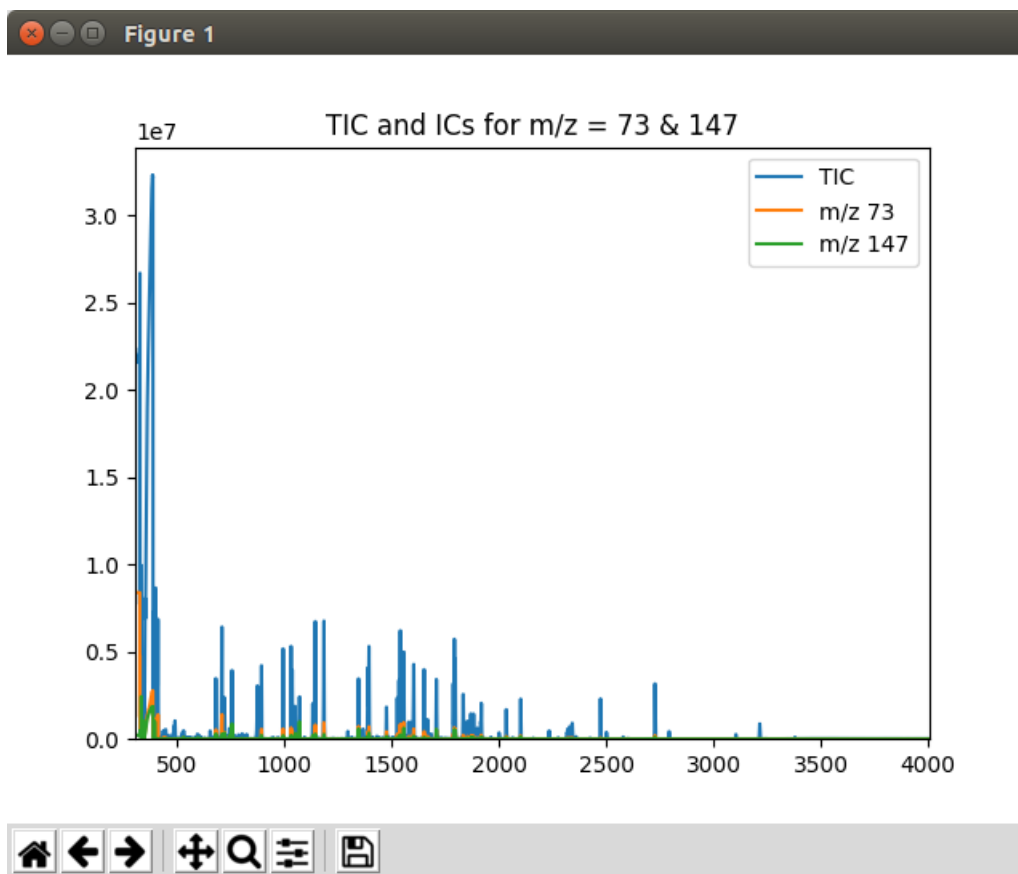


Fig. 2: Graphics window displayed by the script 70b/proc.py

Note: This example is in `pyms-demo/jupyter/Displaying_Multiple_IC.ipynb` and `pyms-demo/70b/proc.py`.

Example: Displaying a Mass Spectrum

The `pyms Display` module can also be used to display individual mass spectra.

To start, load a datafile and create an *IntensityMatrix* as before.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.GCMS.IO.JCAMP import JCAMP_reader
from pyms.IntensityMatrix import build_intensity_matrix_i

jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
tic = data.tic
im = build_intensity_matrix_i(data)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
```

Extract the desired *MassSpectrum* from the *IntensityMatrix*.

```
In [2]: ms = im.get_ms_at_index(1024)
```

Import `matplotlib` and the `lplot_mass_spec()` function, create a subplot, and plot the spectrum on the chart:

```
In [3]: import matplotlib.pyplot as plt
from pyms.Display import plot_mass_spec

%matplotlib inline
# Change to ``notebook`` for an interactive view

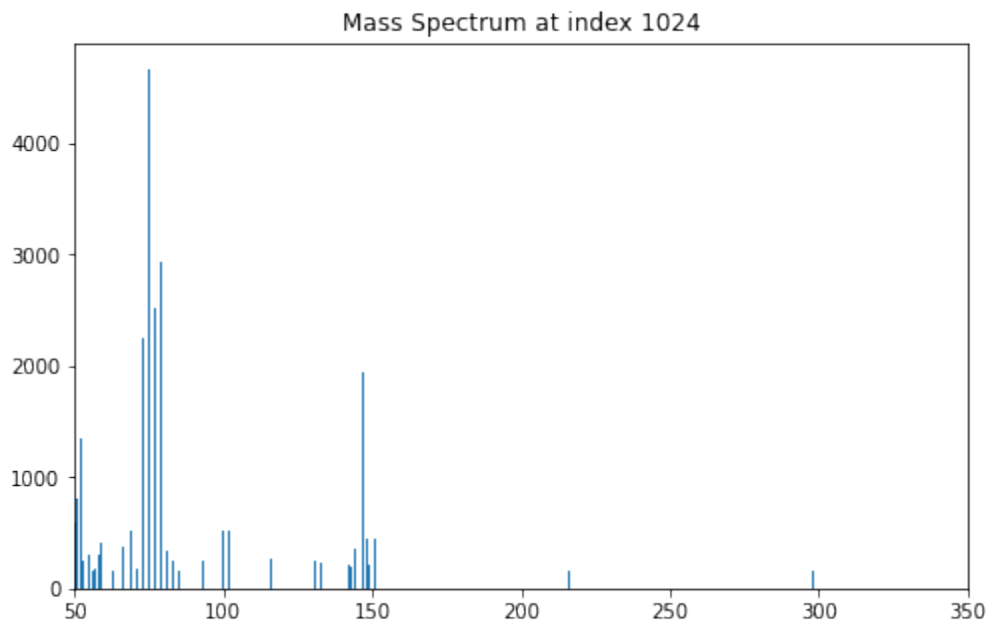
fig, ax = plt.subplots(1, 1, figsize=(8, 5))

# Plot the spectrum
plot_mass_spec(ax, ms)

# Set the title
ax.set_title("Mass Spectrum at index 1024")

# Reduce the x-axis range to better visualise the data
ax.set_xlim(50, 350)

plt.show()
```



When not running in Jupyter Notebook, the spectrum may appear in a separate window looking like this:

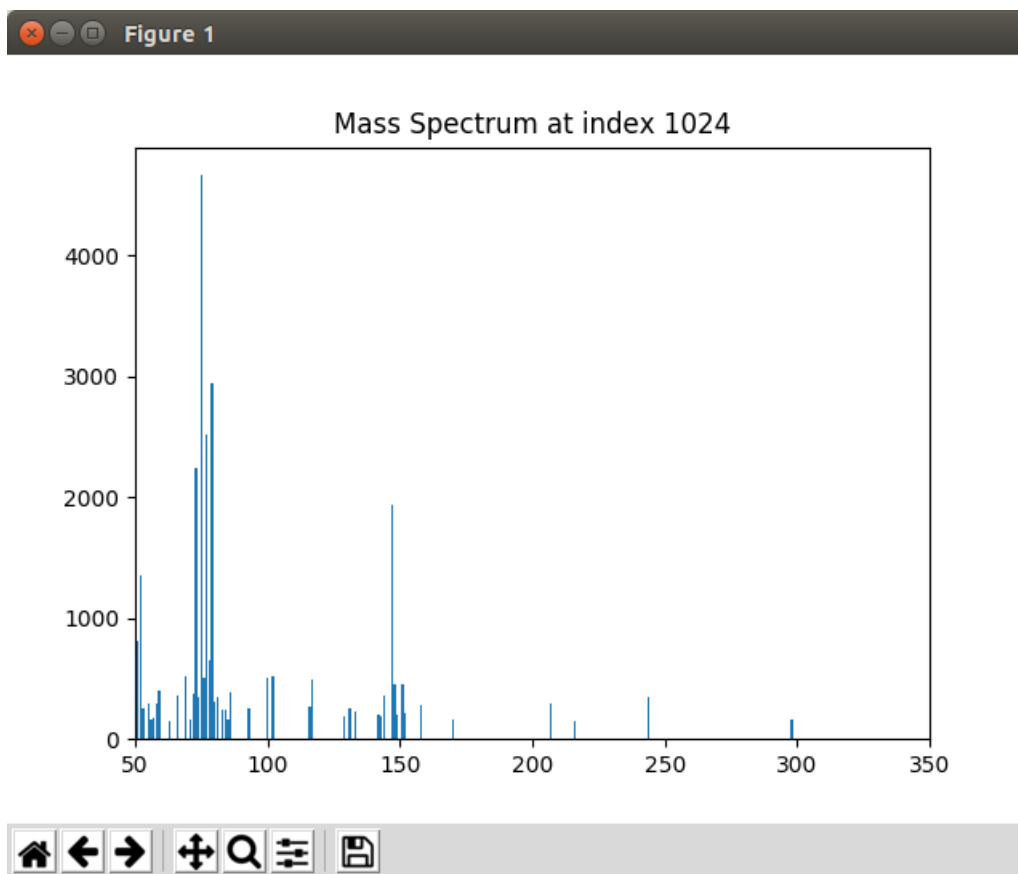


Fig. 3: Graphics window displayed by the script 70c/proc.py

Note: This example is in `pyms-demo/jupyter/Displaying_Mass_Spec.ipynb` and `pyms-demo/70c/proc.py`.

Example: Displaying Detected Peaks

The `pyms.Display.Display` module also allows for detected peaks to be marked on a TIC plot.

First, setup the paths to the datafiles and the output directory, then import JCAMP_reader and build_intensity_matrix.

```
In [1]: import pathlib
data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"

from pyms.GCMS.IO.JCAMP import JCAMP_reader
from pyms.IntensityMatrix import build_intensity_matrix
```

Read the raw data files, extract the TIC and build the *IntensityMatrix*.

```
In [2]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
data.trim("500s", "2000s")
tic = data.tic
im = build_intensity_matrix(data)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
Trimming data to between 520 and 4517 scans
```

Perform pre-filtering and peak detection. For more information on detecting peaks see “Peak detection and representation <chapter06.html>_”.

```
In [3]: from pyms.Noise.SavitzkyGolay import savitzky_golay
from pyms.TopHat import tophat
from pyms.BillerBiemann import BillerBiemann, rel_threshold, num_ions_threshold

n_scan, n_mz = im.size

for ii in range(n_mz):
    ic = im.get_ic_at_index(ii)
    ic_smooth = savitzky_golay(ic)
    ic_bc = tophat(ic_smooth, struct="1.5m")
    im.set_ic_at_index(ii, ic_bc)

# Detect Peaks
peak_list = BillerBiemann(im, points=9, scans=2)

print("Number of peaks found: ", len(peak_list))

# Filter the peak list, first by removing all intensities in a peak less than a
# given relative threshold, then by removing all peaks that have less than a
# given number of ions above a given value

pl = rel_threshold(peak_list, percent=2)
```

(continues on next page)

(continued from previous page)

```
new_peak_list = num_ions_threshold(pl, n=3, cutoff=10000)

print("Number of filtered peaks: ", len(new_peak_list))
```

```
Number of peaks found: 1467
Number of filtered peaks: 72
```

Get Ion Chromatograms for 4 separate m/z channels.

```
In [4]: ic191 = im.get_ic_at_mass(191)
        ic73 = im.get_ic_at_mass(73)
        ic57 = im.get_ic_at_mass(57)
        ic55 = im.get_ic_at_mass(55)
```

Import matplotlib, and the `plot_ic()` and `plot_peaks()` functions.

```
In [5]: import matplotlib.pyplot as plt
        from pyms.Display import plot_ic, plot_peaks
```

Create a subplot, and plot the TIC.

```
In [6]: %matplotlib inline
        # Change to ``notebook`` for an interactive view

        fig, ax = plt.subplots(1, 1, figsize=(8, 5))

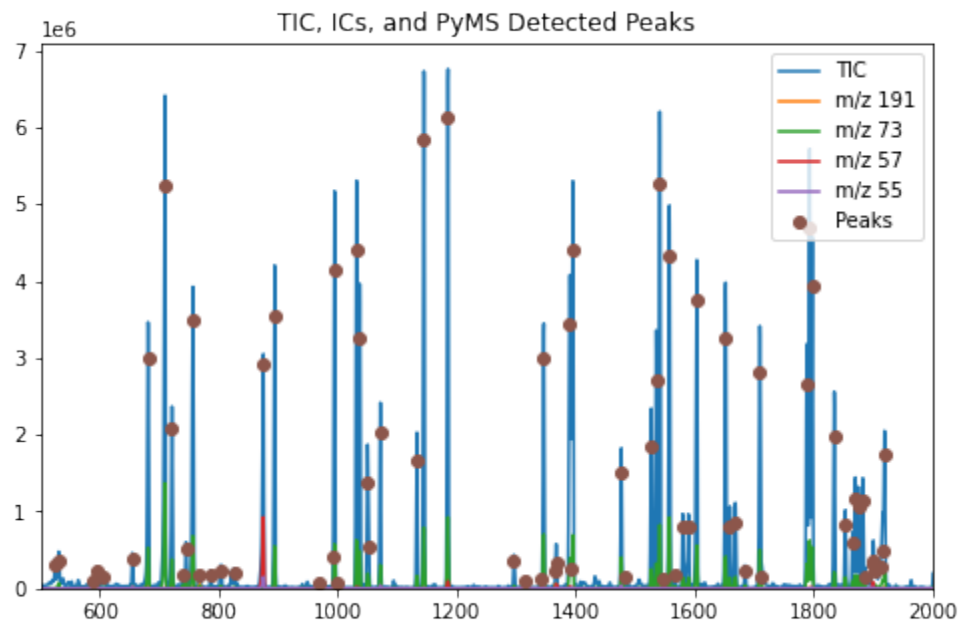
        # Plot the ICs
        plot_ic(ax, tic, label="TIC")
        plot_ic(ax, ic191, label="m/z 191")
        plot_ic(ax, ic73, label="m/z 73")
        plot_ic(ax, ic57, label="m/z 57")
        plot_ic(ax, ic55, label="m/z 55")

        # Plot the peaks
        plot_peaks(ax, new_peak_list)

        # Set the title
        ax.set_title('TIC, ICs, and PyMS Detected Peaks')

        # Add the legend
        plt.legend()

        plt.show()
```



The function `plot_peaks()` adds the PyMassSpec detected peaks to the figure.

The function `store_peaks()` in `proc_save_peaks.py` stores the peaks, while `load_peaks()` in `proc.py` loads them for the Display class to use.

When not running in Jupyter Notebook, the plot may appear in a separate window looking like this:

Note: This example is in `pyms-demo/jupyter/Displaying_Detected_Peaks.ipynb` and `pyms-demo/71/proc.py`.

Example: User Interaction With The Plot Window

The class `pyms.Display.ClickEventHandler` allows for additional interaction with the plot on top of that provided by matplotlib.

Note: This may not work in Jupyter Notebook

To use the class, first import and process the data before:

```
In [1]: import pathlib

import matplotlib.pyplot as plt

from pyms.GCMS.IO.JCAMP import JCAMP_reader
from pyms.IntensityMatrix import build_intensity_matrix
from pyms.Display import plot_ic, plot_peaks
from pyms.Noise.SavitzkyGolay import savitzky_golay
from pyms.TopHat import tophat
from pyms.BillerBiemann import BillerBiemann, rel_threshold, num_ions_threshold
```

```
In [2]: data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
# Change this if the data files are stored in a different location

output_directory = pathlib.Path(".").resolve() / "output"
```

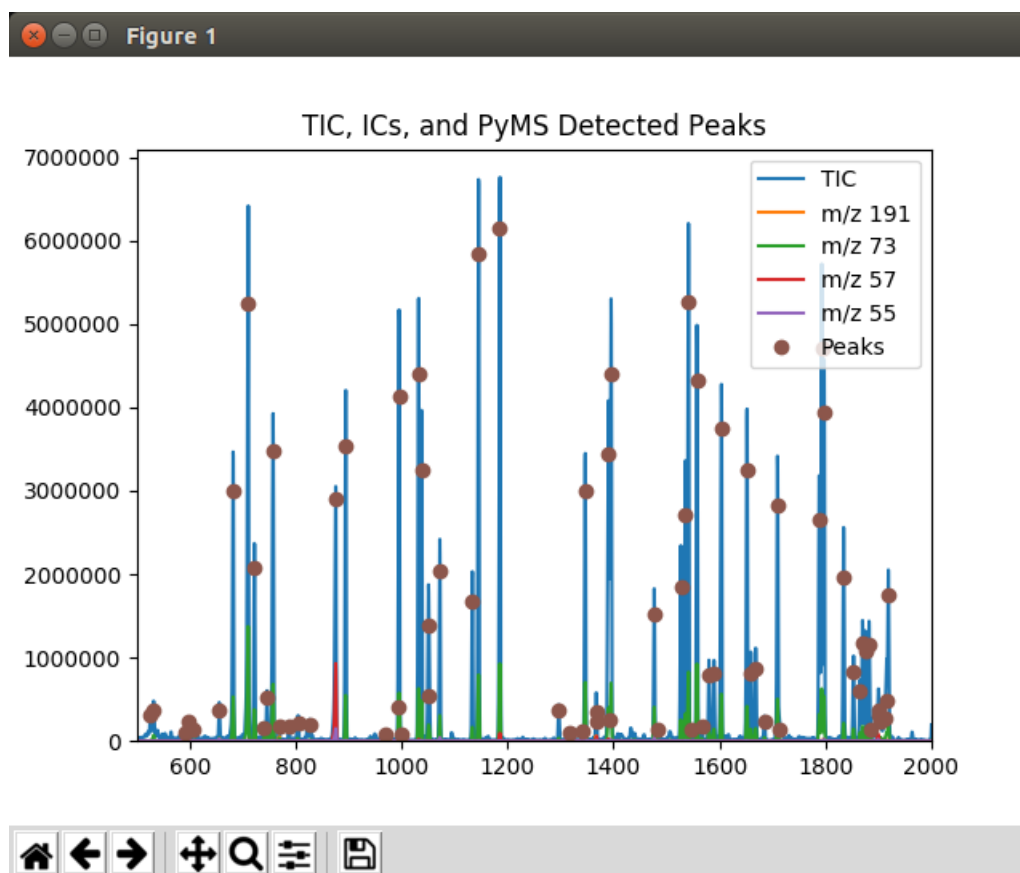



Fig. 4: Graphics window displayed by the script 71/proc.py

```
In [3]: jcamp_file = data_directory / "gc01_0812_066.jdx"
data = JCAMP_reader(jcamp_file)
data.trim("500s", "2000s")
tic = data.tic
im = build_intensity_matrix(data)
```

```
-> Reading JCAMP file '/home/vagrant/PyMassSpec/pyms-data/gc01_0812_066.jdx'
Trimming data to between 520 and 4517 scans
```

```
In [4]: n_scan, n_mz = im.size

for ii in range(n_mz):
    ic = im.get_ic_at_index(ii)
    ic_smooth = savitzky_golay(ic)
    ic_bc = tophat(ic_smooth, struct="1.5m")
    im.set_ic_at_index(ii, ic_bc)
```

```
In [5]: peak_list = BillerBiemann(im, points=9, scans=2)
pl = rel_threshold(peak_list, percent=2)
new_peak_list = num_ions_threshold(pl, n=3, cutoff=10000)

print("Number of filtered peaks: ", len(new_peak_list))
```

```
Number of filtered peaks: 72
```

Creating the plot proceeds much as before, except that `pyms.Display.ClickEventHandler` must be called before `plt.show()`.

You should also assign this to a variable to prevent it being garbage collected.

```
In [6]: from pyms.Display import ClickEventHandler

%matplotlib inline
# Change to ``notebook`` for an interactive view

fig, ax = plt.subplots(1, 1, figsize=(8, 5))

# Plot the TIC
plot_ic(ax, tic, label="TIC")

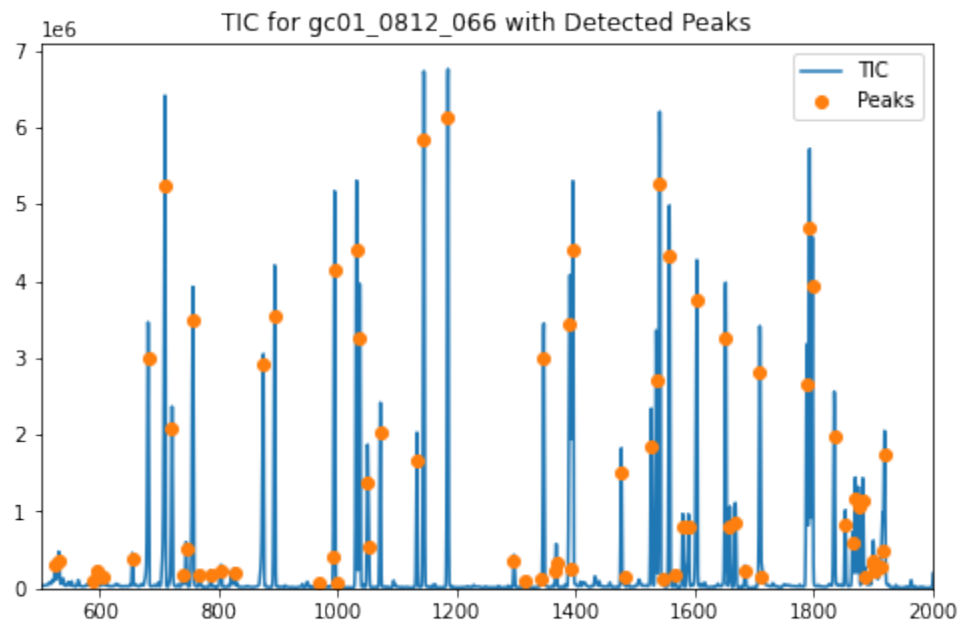
# Plot the peaks
plot_peaks(ax, new_peak_list)

# Set the title
ax.set_title('TIC for gc01_0812_066 with Detected Peaks')

# Set up the ClickEventHandler
handler = ClickEventHandler(new_peak_list)

# Add the legend
plt.legend()

plt.show()
```



Clicking on a Peak causes a list of the 5 highest intensity ions at that Peak to be written to the terminal in order. The output should look similar to this:



```
RT: 1031.823
Mass      Intensity
158.0     2206317.857142857
73.0      628007.1428571426
218.0     492717.04761904746
159.0     316150.4285714285
147.0     196663.95238095228
```


If there is no Peak close to the point on the chart that was clicked, the the following will be shown in the terminal:

```
No Peak at this point
```

The `pym.s.Display.ClickEventHandler` class can be configured with a different tolerance, in seconds, when clicking on a Peak, and to display a different number of top n ions when a Peak is clicked.

In addition, clicking the right mouse button on a Peak displays the mass spectrum at the peak in a new window.

To zoom in on a portion of the plot, select the  button, hold down the left mouse button while dragging a rectangle over the area of interest. To return to the original view, click on the  button.

The  button allows panning across the zoomed plot.

Note: This example is in `pym.s-demo/jupyter/Display_User_Interaction.ipynb` and `pym.s-demo/72/proc.py`.

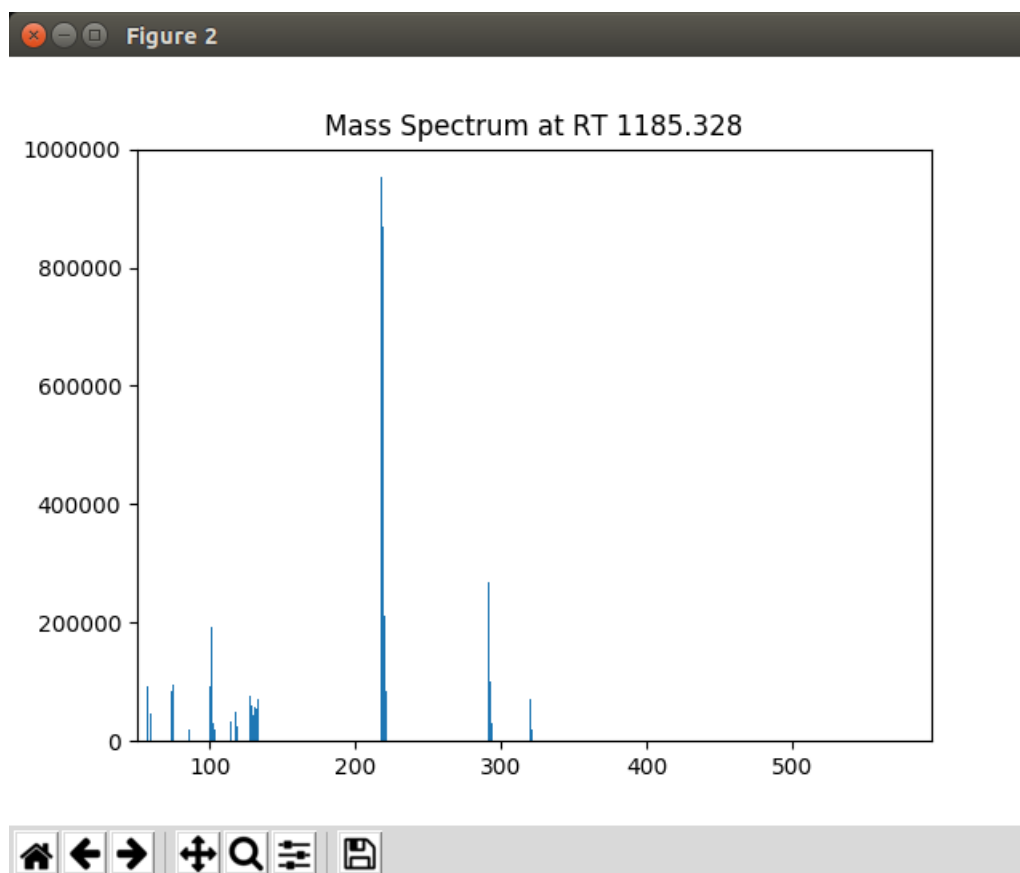


Fig. 5: The mass spectrum displayed by PyMassSpec when a peak in the graphics window is right clicked

6.3 Documentation

6.3.1 `pym.s.Base`

Base for PyMassSpec classes.

Classes:

<code>pym.sBaseClass()</code>	Base class.
-------------------------------	-------------

class `pym.sBaseClass`

Bases: `object`

Base class.

Methods:

<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
--	--

dump (*file_name*, *protocol*=3)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – Filename to save the dump as.
- **protocol** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

6.3.2 `pym.sBillerBiemann`

Functions to perform Biller and Biemann deconvolution.

Functions:

<code>BillerBiemann(im[, points, scans])</code>	Deconvolution based on the algorithm of Biller and Biemann (1974).
<code>get_maxima_indices(ion_intensities[, points])</code>	Returns the scan indices for the apexes of the ion.
<code>get_maxima_list(ic[, points])</code>	List of retention time and intensity of local maxima for ion.
<code>get_maxima_list_reduced(ic, mp_rt[, points, ...])</code>	List of retention time and intensity of local maxima for ion.
<code>get_maxima_matrix(im[, points, scans])</code>	Constructs a matrix containing only data for scans in which particular ions apexed.
<code>num_ions_threshold(pl, n, cutoff[, copy_peaks])</code>	Remove Peaks where there are fewer than <i>n</i> ions with intensities above the given threshold.
<code>rel_threshold(pl[, percent, copy_peaks])</code>	Remove ions with relative intensities less than the given relative percentage of the maximum intensity.
<code>sum_maxima(im[, points, scans])</code>	Reconstruct the TIC as sum of maxima.

BillerBiemann (*im*, *points=3*, *scans=1*)

Deconvolution based on the algorithm of Biller and Biemann (1974).

Parameters

- **im** (*BaseIntensityMatrix*)
- **points** (*int*) – Number of scans over which to consider a maxima to be a peak. Default 3.
- **scans** (*int*) – Number of scans to combine peaks from to compensate for spectra skewing. Default 1.

Return type `List[Peak]`

Returns List of detected peaks

Authors Andrew Isaac, Dominic Davis-Foster (type assertions)

get_maxima_indices (*ion_intensities*, *points=3*)

Returns the scan indices for the apexes of the ion.

Parameters

- **ion_intensities** (`Union[Sequence, ndarray]`) – A list of intensities for a single ion.
- **points** (*int*) – Number of scans over which to consider a maxima to be a peak. Default 3.

Author Andrew Isaac, Dominic Davis-Foster (type assertions)

Example:

```
>>> # A trivial set of data with two clear peaks
>>> data = [1, 2, 3, 4, 5, 4, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1]
>>> get_maxima_indices(data)
[4, 13]
>>> # Wider window (more points)
>>> get_maxima_indices(data, points=10)
[13]
```

Return type `List[int]`

get_maxima_list (*ic*, *points=3*)

List of retention time and intensity of local maxima for ion.

Parameters

- **ic** (*IonChromatogram*)
- **points** (*int*) – Number of scans over which to consider a maxima to be a peak. Default 3.

Return type `List[List[float]]`

Returns A list of retention time and intensity of local maxima for ion.

Author Andrew Isaac, Dominic Davis-Foster (type assertions)

get_maxima_list_reduced (*ic*, *mp_rt*, *points=13*, *window=3*)

List of retention time and intensity of local maxima for ion.

Only peaks around a specific retention time are recorded.

Created for use with gap filling algorithm.

Parameters

- **ic** (*IonChromatogram*)
- **mp_rt** (*float*) – The retention time of the missing peak
- **points** (*int*) – Number of scans over which to consider a maxima to be a peak. Default 13.
- **window** (*int*) – The window around mp_rt where peaks should be recorded. Default 3.

Return type `List[Tuple[float, float]]`

Returns A list of 2-element tuple containing the retention time and intensity of local maxima for each ion.

Author Andrew Isaac, Dominic Davis-Foster (type assertions)

get_maxima_matrix (*im, points=3, scans=1*)

Constructs a matrix containing only data for scans in which particular ions apexed.

The data can be optionally consolidated into the scan within a range with the highest total intensity by adjusting the *scans* parameter. By default this is 1, which does not consolidate the data.

The columns are ion masses and the rows are scans. Get matrix of local maxima for each ion.

Parameters

- **im** (*BaseIntensityMatrix*)
- **points** (*int*) – Number of scans over which to consider a maxima to be a peak. Default 3.
- **scans** (*int*) – Number of scans to combine peaks from to compensate for spectra skewing. Default 1.

Return type `ndarray`

Returns A matrix of giving the intensities of ion masses (columns) and for each scan (rows).

Author Andrew Isaac, Dominic Davis-Foster (type assertions)

num_ions_threshold (*pl, n, cutoff, copy_peaks=True*)

Remove Peaks where there are fewer than n ions with intensities above the given threshold.

Parameters

- **pl** (*Sequence[Peak]*)
- **n** (*int*) – Minimum number of ions that must have intensities above the cutoff.
- **cutoff** (*float*) – The minimum intensity threshold.
- **copy_peaks** (*bool*) – Whether the returned peak list should contain copies of the peaks. Default `True`.

Return type `List[Peak]`

Returns A new list of Peak objects.

Author Andrew Isaac, Dominic Davis-Foster (type assertions)

rel_threshold (*pl*, *percent*=2, *copy_peaks*=True)

Remove ions with relative intensities less than the given relative percentage of the maximum intensity.

Parameters

- **pl** (*Sequence*[*Peak*])
- **percent** (*float*) – Threshold for relative percentage of intensity. Default 2%.
- **copy_peaks** (*bool*) – Whether the returned peak list should contain copies of the peaks. Default True.

Return type *List*[*Peak*]

Returns A new list of Peak objects with threshold ions.

Author Andrew Isaac, Dominic Davis-Foster (type assertions)

sum_maxima (*im*, *points*=3, *scans*=1)

Reconstruct the TIC as sum of maxima.

Parameters

- **im** (*BaseIntensityMatrix*)
- **points** (*int*) – Peak if maxima over ‘points’ number of scans. Default 3.
- **scans** (*int*) – Number of scans to combine peaks from to compensate for spectra skewing. Default 1.

Return type *IonChromatogram*

Returns The reconstructed TIC.

Author Andrew Isaac, Dominic Davis-Foster (type assertions)

6.3.3 `pyms.Display`

Class to Display Ion Chromatograms and TIC.

Classes:

<code>ClickEventHandler</code> (<i>peak_list</i> [, <i>fig</i> , <i>ax</i> , ...])	Class to enable clicking of chromatogram to view the intensities top n most intense ions at that peak, and viewing of the mass spectrum with a right click
<code>Display</code> ([<i>fig</i> , <i>ax</i>])	Class to display Ion Chromatograms and Total Ion Chromatograms from <code>pyms.IonChromatogram</code> . <code>IonChromatogram</code> using <code>matplotlib.pyplot</code> .

Functions:

<code>invert_mass_spec</code> (<i>mass_spec</i> [, <i>inplace</i>])	Invert the mass spectrum for display in a head2tail plot.
<code>plot_head2tail</code> (<i>ax</i> , <i>top_mass_spec</i> , ...[, ...])	Plots two mass spectra head to tail.
<code>plot_ic</code> (<i>ax</i> , <i>ic</i> [, <i>minutes</i>])	Plots an Ion Chromatogram.
<code>plot_mass_spec</code> (<i>ax</i> , <i>mass_spec</i> , ** <i>kwargs</i>)	Plots a Mass Spectrum.
<code>plot_peaks</code> (<i>ax</i> , <i>peak_list</i> [, <i>label</i> , <i>style</i>])	Plots the locations of peaks as found by PyMassSpec.

class ClickEventHandler (*peak_list*, *fig*=None, *ax*=None, *tolerance*=0.005, *n_intensities*=5)

Bases: `object`

Class to enable clicking of chromatogram to view the intensities top n most intense ions at that peak, and viewing of the mass spectrum with a right click

Methods:

<code>get_n_largest(intensity_list)</code>	Computes the indices of the largest n ion intensities for writing to console.
<code>onclick(event)</code>	Finds the n highest intensity m/z channels for the selected peak.

get_n_largest (*intensity_list*)

Computes the indices of the largest n ion intensities for writing to console.

Parameters *intensity_list* (`List[float]`) – List of Ion intensities

Return type `List[int]`

Returns Indices of largest n ion intensities

onclick (*event*)

Finds the n highest intensity m/z channels for the selected peak. The peak is selected by clicking on it. If a button other than the left one is clicked, a new plot of the mass spectrum is displayed.

Parameters *event* – a mouse click by the user

class Display (*fig=None, ax=None*)

Bases: `object`

Class to display Ion Chromatograms and Total Ion Chromatograms from `pymms.IonChromatogram.IonChromatogram` using `matplotlib.pyplot`.

Parameters

- **fig** (`Figure`) – figure object to use. Default `None`.
- **ax** (`Axes`) – axes object to use. Default `None`.

If *fig* is not given then *fig* and *ax* default to:

```
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
```

If only *fig* is given then *ax* defaults to:

```
>>> ax = fig.add_subplot(111)
```

Author Sean O'Callaghan

Author Vladimir Likic

Author Dominic Davis-Foster

Methods:

<code>do_plotting([plot_label])</code>	Plots TIC and IC(s) if they have been created by <code>plot_tic()</code> or <code>plot_ics()</code> .
<code>get_5_largest(intensity_list)</code>	Returns the indices of the 5 largest ion intensities.
<code>onclick(event)</code>	Finds the 5 highest intensity m/z channels for the selected peak.
<code>plot_ic(ic, **kwargs)</code>	Plots an Ion Chromatogram.
<code>plot_mass_spec(mass_spec, **kwargs)</code>	Plots a Mass Spectrum.
<code>plot_peaks(peak_list[, label])</code>	Plots the locations of peaks as found by Py-MassSpec.
<code>plot_tic(tic[, minutes])</code>	Plots a Total Ion Chromatogram.
<code>save_chart(filepath[, filetypes])</code>	Save the chart to the given path with the given filetypes.
<code>show_chart()</code>	Show the chart on screen.

do_plotting (*plot_label=None*)

Plots TIC and IC(s) if they have been created by `plot_tic()` or `plot_ics()`.

Also adds detected peaks if they have been added by `plot_peaks()`

Parameters `plot_label` (Optional[str]) – Label for the plot to show e.g. the data origin. Default None.

static get_5_largest (*intensity_list*)

Returns the indices of the 5 largest ion intensities.

Parameters `intensity_list` (List[float]) – List of Ion intensities

Return type List[int]

onclick (*event*)

Finds the 5 highest intensity m/z channels for the selected peak. The peak is selected by clicking on it. If a button other than the left one is clicked, a new plot of the mass spectrum is displayed.

Parameters `event` – a mouse click by the user

plot_ic (*ic, **kwargs*)

Plots an Ion Chromatogram.

Parameters `ic` (IonChromatogram) – Ion Chromatograms m/z channels for plotting

Other Parameters `matplotlib.lines.Line2D` properties. Used to specify properties like a line label (for auto legends), linewidth, antialiasing, marker face color.

Example:

```
>>> plot_ic(im.get_ic_at_index(5), label='IC @ Index 5', linewidth=2)
```

See https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.lines.Line2D.html

for the list of possible kwargs

Return type List[Line2D]

plot_mass_spec (*mass_spec, **kwargs*)

Plots a Mass Spectrum.

Parameters `mass_spec` (*MassSpectrum*) – The mass spectrum at a given time/index

Other Parameters `matplotlib.lines.Line2D` properties. Used to specify properties like a line label (for auto legends), linewidth, antialiasing, marker face color.

Example:

```
>>> plot_mass_spec(im.get_ms_at_index(5), linewidth=2)
>>> ax.set_title(f"Mass spec for peak at time {im.get_time_at_index(5):5.
↪2f}")
```

See https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.lines.Line2D.html for

the list of possible kwargs

Return type `BarContainer`

plot_peaks (*peak_list*, *label*='Peaks')

Plots the locations of peaks as found by PyMassSpec.

Parameters

- **peak_list** (`List[Peak]`) – List of peaks to plot
- **label** (`str`) – label for plot legend. Default 'Peaks'.

Return type `List[Line2D]`

plot_tic (*tic*, *minutes*=False, ***kwargs*)

Plots a Total Ion Chromatogram.

Parameters

- **tic** (*IonChromatogram*) – Total Ion Chromatogram.
- **minutes** (`bool`) – Whether to show the time in minutes. Default False.

Other Parameters `matplotlib.lines.Line2D` properties. Used to specify properties like a line label (for auto legends), linewidth, antialiasing, marker face color.

Example:

```
>>> plot_tic(data.tic, label='TIC', linewidth=2)
```

See https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.lines.Line2D.html for

the list of possible kwargs

Return type `List[Line2D]`

save_chart (*filepath*, *filetypes*=None)

Save the chart to the given path with the given filetypes.

Parameters

- **filepath** (`str`) – Path and filename to save the chart as. Should not include extension.
- **filetypes** (`Optional[List[str]]`) – List of filetypes to use. Default None.

Author Dominic Davis-Foster

show_chart ()

Show the chart on screen.

Author Dominic Davis-Foster

invert_mass_spec (*mass_spec*, *inplace=False*)

Invert the mass spectrum for display in a head2tail plot.

Parameters

- **mass_spec** (*MassSpectrum*) – The Mass Spectrum to normalize
- **inplace** (*bool*) – Whether the inversion should be applied to the *MassSpectrum* object given, or to a copy (default behaviour). Default *False*.

Return type *MassSpectrum*

Returns The normalized mass spectrum

plot_head2tail (*ax*, *top_mass_spec*, *bottom_mass_spec*, *top_spec_kwargs=None*,
bottom_spec_kwargs=None)

Plots two mass spectra head to tail.

Parameters

- **ax** (*Axes*) – The axes to plot the MassSpectra on
- **top_mass_spec** (*MassSpectrum*) – The Mass Spectrum to plot on top
- **bottom_mass_spec** (*MassSpectrum*) – The Mass Spectrum to plot on the bottom
- **top_spec_kwargs** (*Optional[Dict]*) – A dictionary of keyword arguments for the top mass spectrum. Defaults to red with a line width of 0.5
- **bottom_spec_kwargs** (*Optional[Dict]*) – A dictionary of keyword arguments for the bottom mass spectrum. Defaults to blue with a line width of 0.5

top_spec_kwargs and *bottom_spec_kwargs* are used to specify properties like a line label (for auto legends), linewidth, antialiasing, marker face color.

See https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.lines.Line2D.html for the list of possible kwargs

Returns A tuple of container with all the bars and optionally errorbars for the top and bottom spectra.

Return type tuple of *matplotlib.container.BarContainer*

plot_ic (*ax*, *ic*, *minutes=False*, ***kwargs*)

Plots an Ion Chromatogram.

Parameters

- **ax** (*Axes*) – The axes to plot the IonChromatogram on
- **ic** (*IonChromatogram*) – Ion Chromatograms m/z channels for plotting
- **minutes** (*bool*) – Whether the x-axis should be plotted in minutes. Default *False* (plotted in seconds). Default *False*.

Other Parameters *matplotlib.lines.Line2D* properties. Used to specify properties like a line label (for auto legends), linewidth, antialiasing, marker face color.

Example:

```
>>> plot_ic(im.get_ic_at_index(5), label='IC @ Index 5', linewidth=2)
```

See

https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.lines.Line2D.html
for the list of possible kwargs

Return type `List[Line2D]`

Returns A list of Line2D objects representing the plotted data.

plot_mass_spec (*ax*, *mass_spec*, ***kwargs*)

Plots a Mass Spectrum.

Parameters

- **ax** (*Axes*) – The axes to plot the MassSpectrum on
- **mass_spec** (*MassSpectrum*) – The mass spectrum to plot

Other Parameters `matplotlib.lines.Line2D` properties. Used to specify properties like a line label (for auto legends), linewidth, antialiasing, marker face color.

Example:

```
>>> plot_mass_spec(im.get_ms_at_index(5), linewidth=2)
>>> ax.set_title(f"Mass spec for peak at time {im.get_time_at_index(5):5.2f}")
```

See

https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.lines.Line2D.html
for the list of possible kwargs

Returns Container with all the bars and optionally errorbars.

Return type `matplotlib.container.BarContainer`

plot_peaks (*ax*, *peak_list*, *label='Peaks'*, *style='o'*)

Plots the locations of peaks as found by PyMassSpec.

Parameters

- **ax** (*Axes*) – The axes to plot the peaks on
- **peak_list** (`List[Peak]`) – List of peaks to plot
- **label** (*str*) – label for plot legend. Default 'Peaks'.
- **style** (*str*) – The marker style. See https://matplotlib.org/3.1.1/api/markers_api.html for a complete list. Default 'o'.

Return type `List[Line2D]`

Returns A list of Line2D objects representing the plotted data.

6.3.4 `pyms.DPA`

Table of Contents

- `pyms.DPA`
 - `pyms.DPA.Alignment`
 - `pyms.DPA.PairwiseAlignment`
 - `pyms.DPA.IO`
 - `pyms.DPA.clustering`

Alignment of peak lists by dynamic programming.

`pyms.DPA.Alignment`

Classes for peak alignment by dynamic programming.

Classes:

<code>Alignment(expr)</code>	Models an alignment of peak lists.
------------------------------	------------------------------------

Functions:

<code>exprl2alignment(expr_list)</code>	Converts a list of experiments into a list of alignments.
---	---

class `Alignment` (*expr*)

Bases: `object`

Models an alignment of peak lists.

Parameters `expr` (`Optional[Experiment]`) – The experiment to be converted into an alignment object.

Authors Woon Wai Keen, Qiao Wang, Vladimir Likic, Dominic Davis-Foster.

Methods:

<code>__len__()</code>	Returns the length of the alignment, defined as the number of peak positions in the alignment.
<code>aligned_peaks([minutes])</code>	Returns a list of Peak objects where each peak has the combined spectra and average retention time of all peaks that aligned.
<code>common_ion()</code>	Calculates a common ion among the peaks of an aligned peak.
<code>filter_min_peaks(min_peaks)</code>	Filters alignment positions that have less peaks than <code>min_peaks</code> .
<code>get_area_alignment([require_all_expr])</code>	Returns a Pandas dataframe containing the peak areas of the aligned peaks.
<code>get_highest_mz_ion(ion_dict)</code>	Returns the preferred ion for quantitation.
<code>get_ms_alignment([require_all_expr])</code>	Returns a Pandas dataframe of mass spectra for the aligned peaks.

continues on next page

Table 10 – continued from previous page

<code>get_peak_alignment([minutes, require_all_expr])</code>	re-	Returns a Pandas dataframe of aligned retention times.
<code>get_peaks_alignment([require_all_expr])</code>		Returns a Pandas dataframe of Peak objects for the aligned peaks.
<code>write_common_ion_csv(area_file_name, ..., ...)</code>		Writes the alignment to CSV files.
<code>write_csv(rt_file_name, area_file_name[, ...])</code>		Writes the alignment to CSV files.
<code>write_ion_areas_csv(ms_file_name[, minutes])</code>	min-	Write Ion Areas to CSV File.

Attributes:

<code>expr_code</code>	List of experiment codes.
<code>peakalgt</code>	
<code>peakpos</code>	
<code>similarity</code>	

__len__()

Returns the length of the alignment, defined as the number of peak positions in the alignment.

Return type `int`

Authors Qiao Wang, Vladimir Likic

aligned_peaks (*minutes=False*)

Returns a list of Peak objects where each peak has the combined spectra and average retention time of all peaks that aligned.

Parameters **minutes** (`bool`) – Whether retention times are in minutes. If `False`, retention time are in seconds. Default `False`.

Return type `Sequence[Optional[Peak]]`

Returns A list of composite peaks based on the alignment.

Author Andrew Isaac

common_ion()

Calculates a common ion among the peaks of an aligned peak.

Return type `List[float]`

Returns A list of the highest intensity common ion for all aligned peaks.

Author Sean O’Callaghan

expr_code

Type: `List[str]`

List of experiment codes.

filter_min_peaks (*min_peaks*)

Filters alignment positions that have less peaks than `min_peaks`.

This function is useful only for within state alignment.

Parameters `min_peaks` (`int`) – Minimum number of peaks required for the alignment position to survive filtering.

Author Qiao Wang

get_area_alignment (`require_all_expr=True`)

Returns a Pandas dataframe containing the peak areas of the aligned peaks.

Parameters `require_all_expr` (`bool`) – Whether the peak must be present in all experiments to be included in the data frame. Default `True`.

Authors Woon Wai Keen, Andrew Isaac, Vladimir Likic, Dominic Davis-Foster

Return type `DataFrame`

static get_highest_mz_ion (`ion_dict`)

Returns the preferred ion for quantitation.

Looks at the list of candidate ions, selects those which have highest occurrence, and selects the heaviest of those.

Parameters `ion_dict` (`Dict[float, int]`) – a dictionary of m/z value: number of occurrences.

Return ion The ion with the highest m/z value.

Return type `float`

get_ms_alignment (`require_all_expr=True`)

Returns a Pandas dataframe of mass spectra for the aligned peaks.

Parameters `require_all_expr` (`bool`) – Whether the peak must be present in all experiments to be included in the data frame. Default `True`.

Authors Woon Wai Keen, Andrew Isaac, Vladimir Likic, Dominic Davis-Foster

Return type `DataFrame`

get_peak_alignment (`minutes=True`, `require_all_expr=True`)

Returns a Pandas dataframe of aligned retention times.

Parameters

- **minutes** (`bool`) – Whether to return retention times in minutes. If `False`, retention time will be returned in seconds. Default `True`.
- **require_all_expr** (`bool`) – Whether the peak must be present in all experiments to be included in the data frame. Default `True`.

Authors Woon Wai Keen, Andrew Isaac, Vladimir Likic, Dominic Davis-Foster

Return type `DataFrame`

get_peaks_alignment (`require_all_expr=True`)

Returns a Pandas dataframe of Peak objects for the aligned peaks.

Parameters `require_all_expr` (`bool`) – Whether the peak must be present in all experiments to be included in the data frame. Default `True`.

Authors Woon Wai Keen, Andrew Isaac, Vladimir Likic, Dominic Davis-Foster

Return type `DataFrame`

peakalgt
Type: `List[List[Peak]]`

peakpos
Type: `List[List[Peak]]`

similarity
Type: `Optional[float]`

write_common_ion_csv (*area_file_name*, *top_ion_list*, *minutes=True*)

Writes the alignment to CSV files.

This function writes two files: one containing the alignment of peak retention times and the other containing the alignment of peak areas.

Parameters

- **area_file_name** (`Union[str, Path, PathLike]`) – The name for the areas alignment file.
- **top_ion_list** (`Sequence[float]`) – A list of the highest intensity common ion along the aligned peaks.
- **minutes** (`bool`) – Whether to save retention times in minutes. If `False`, retention time will be saved in seconds. Default `True`.

Authors Woon Wai Keen, Andrew Isaac, Sean O’Callaghan, Vladimir Likic, Dominic Davis-Foster (pathlib support)

write_csv (*rt_file_name*, *area_file_name*, *minutes=True*)

Writes the alignment to CSV files.

This function writes two files: one containing the alignment of peak retention times and the other containing the alignment of peak areas.

Parameters

- **rt_file_name** (`Union[str, Path, PathLike]`) – The name for the retention time alignment file.
- **area_file_name** (`Union[str, Path, PathLike]`) – The name for the areas alignment file.
- **minutes** (`bool`) – Whether to save retention times in minutes. If `False`, retention time will be saved in seconds. Default `True`.

Authors Woon Wai Keen, Andrew Isaac, Vladimir Likic, David Kainer, Dominic Davis-Foster (pathlib support)

write_ion_areas_csv (*ms_file_name*, *minutes=True*)

Write Ion Areas to CSV File.

Parameters

- **ms_file_name** (`Union[str, Path, PathLike]`) – The name of the file
- **minutes** (`bool`) – Whether to save retention times in minutes. If `False`, retention time will be saved in seconds. Default `True`.

Authors David Kainer, Dominic Davis-Foster (pathlib support)

exprl2alignment (*expr_list*)

Converts a list of experiments into a list of alignments.

Parameters **expr_list** (*List[Experiment]*) – The list of experiments to be converted into an alignment objects.

Return type *List[Alignment]*

Returns A list of alignment objects for the experiments.

Author Vladimir Likic

pyms.DPA.PairwiseAlignment

Classes for peak alignment by dynamic programming.

Classes:

<i>PairwiseAlignment</i> (alignments, D, gap)	Models pairwise alignment of alignments.
---	--

Functions:

<i>align</i> (a1, a2, D, gap)	Aligns two alignments.
<i>align_with_tree</i> (T[, min_peaks])	Aligns a list of alignments using the supplied guide tree.
<i>alignment_compare</i> (x, y)	A helper function for sorting peak positions in a alignment.
<i>alignment_similarity</i> (traces, score_matrix, gap)	Calculates similarity score between two alignments (new method).
<i>dp</i> (S, gap_penalty)	Solves optimal path in score matrix based on global sequence alignment.
<i>merge_alignments</i> (A1, A2, traces)	Merges two alignments with gaps added in from DP traceback.
<i>position_similarity</i> (pos1, pos2, D)	Calculates the similarity between the two alignment positions.
<i>score_matrix</i> (a1, a2, D)	Calculates the score matrix between two alignments.
<i>score_matrix_mpi</i> (a1, a2, D)	Calculates the score matrix between two alignments.

class PairwiseAlignment (*alignments, D, gap*)

Bases: *object*

Models pairwise alignment of alignments.

Parameters

- **alignments** (*List[Alignment]*) – A list of alignments.
- **D** (*float*) – Retention time tolerance parameter for pairwise alignments.
- **gap** (*float*) – Gap parameter for pairwise alignments.

Authors Woon Wai Keen, Vladimir Likic

align (*a1, a2, D, gap*)

Aligns two alignments.

Parameters

- **a1** (*Alignment*) – The first alignment
- **a2** (*Alignment*) – The second alignment
- **D** (*float*) – Retention time tolerance
- **gap** (*float*) – Gap penalty

Return type *Alignment*

Returns Aligned alignments

Authors Woon Wai Keen, Vladimir Likic

align_with_tree (*T*, *min_peaks=1*)

Aligns a list of alignments using the supplied guide tree.

Parameters

- **T** (*PairwiseAlignment*) – The pairwise alignment object.
- **min_peaks** (*int*) – Default 1.

Return type *Alignment*

Returns The final alignment consisting of aligned input alignments.

Authors Woon Wai Keen, Vladimir Likic

alignment_compare (*x*, *y*)

A helper function for sorting peak positions in a alignment.

Parameters

- **x**
- **y**

Return type *int*

alignment_similarity (*traces*, *score_matrix*, *gap*)

Calculates similarity score between two alignments (new method).

Parameters

- **traces** – Traceback from DP algorithm.
- **score_matrix** – Score matrix of the two alignments.
- **gap** (*float*) – Gap penalty.

Return type *float*

Returns Similarity score (i.e. more similar => higher score)

Authors Woon Wai Keen, Vladimir Likic

dp (*S*, *gap_penalty*)

Solves optimal path in score matrix based on global sequence alignment.

Parameters

- **S** – Score matrix
- **gap_penalty** (*float*) – Gap penalty

Return type `Dict`

Returns A dictionary of results

Author Tim Erwin

merge_alignments (*A1, A2, traces*)

Merges two alignments with gaps added in from DP traceback.

Parameters

- **A1** (*Alignment*) – First alignment.
- **A2** (*Alignment*) – Second alignment.
- **traces** – DP traceback.

Return type `Alignment`

Returns A single alignment from A1 and A2.

Authors Woon Wai Keen, Vladimir Likic, Qiao Wang

position_similarity (*pos1, pos2, D*)

Calculates the similarity between the two alignment positions.

A score of 0 is best and 1 is worst.

Parameters

- **pos1** – The position of the first alignment.
- **pos2** – The position of the second alignment.
- **D** (`float`) – Retention time tolerance.

Return type `float`

Returns The similarity value for the current position.

Authors Qiao Wang, Vladimir Likic, Andrew Isaac

score_matrix (*a1, a2, D*)

Calculates the score matrix between two alignments.

Parameters

- **a1** (*Alignment*) – The first alignment.
- **a2** (*Alignment*) – The second alignment.
- **D** (`float`) – Retention time tolerance.

Return type `ndarray`

Returns Aligned alignments.

Authors Qiao Wang, Andrew Isaac

score_matrix_mpi (*a1, a2, D*)

Calculates the score matrix between two alignments.

Parameters

- **a1** (*Alignment*) – The first alignment.

- **a2** (*Alignment*) – The second alignment.
- **D** (*float*) – Retention time tolerance.

Return type *Alignment*

Returns Aligned alignments

Authors Qiao Wang, Andrew Isaac

pyms.DPA.IO

Functions for writing peak alignment to various file formats.

Functions:

<code>write_excel(alignment, file_name[, minutes])</code>	Writes the alignment to an excel file, with colouring showing possible mis-alignments.
<code>write_mass_hunter_csv(alignment, file_name, ...)</code>	Creates a csv file with UID, common and qualifying ions and their ratios for mass hunter interpretation.
<code>write_transposed_output(alignment, file_name)</code>	type alignment <i>Alignment</i>

write_excel (*alignment, file_name, minutes=True*)

Writes the alignment to an excel file, with colouring showing possible mis-alignments.

Parameters

- **alignment** (*Alignment*) – *pyms.DPA.Alignment.Alignment* object to write to file.
- **file_name** (*Union[str, Path, PathLike]*) – The name for the retention time alignment file.
- **minutes** (*bool*) – Whether to save retention times in minutes. If *False*, retention time will be saved in seconds. Default *True*.

Author David Kainer

write_mass_hunter_csv (*alignment, file_name, top_ion_list*)

Creates a csv file with UID, common and qualifying ions and their ratios for mass hunter interpretation.

Parameters

- **alignment** (*Alignment*) – alignment object to write to file
- **file_name** (*Union[str, Path, PathLike]*) – name of the output file.
- **top_ion_list** (*List[int]*) – a list of the common ions for each peak in the averaged peak list for the alignment.

write_transposed_output (*alignment, file_name, minutes=True*)

Parameters

- **alignment** (*Alignment*) – *pyms.DPA.Alignment.Alignment* object to write to file
- **file_name** (*Union[str, Path, PathLike]*) – The name of the file

- **minutes** (*bool*) – Default *True*.

`pyms.DPA.clustering`

Provides `Pycluster.treecluster` regardless of which library provides it.

Functions:

<code>treecluster</code> (<i>data</i> [, <i>mask</i> , <i>weight</i> , <i>transpose</i> , ...])	Perform hierarchical clustering, and return a Tree object.
--	--

treecluster (*data*, *mask*=None, *weight*=None, *transpose*=False, *method*='m', *dist*='e',
distancematrix=None)

Perform hierarchical clustering, and return a Tree object.

This function implements the pairwise single, complete, centroid, and average linkage hierarchical clustering methods.

Keyword arguments:

- *data*: *n*rows x *n*columns array containing the data values.
- *mask*: *n*rows x *n*columns array of integers, showing which data are missing. If *mask*[*i*][*j*]==0, then *data*[*i*][*j*] is missing.
- *weight*: the weights to be used when calculating distances.
- *transpose*: - if False, rows are clustered; - if True, columns are clustered.
- *dist*: specifies the distance function to be used: - *dist* == 'e': Euclidean distance - *dist* == 'b': City Block distance - *dist* == 'c': Pearson correlation - *dist* == 'a': absolute value of the correlation - *dist* == 'u': uncentered correlation - *dist* == 'x': absolute uncentered correlation - *dist* == 's': Spearman's rank correlation - *dist* == 'k': Kendall's tau
- *method*: specifies which linkage method is used: - *method* == 's': Single pairwise linkage - *method* == 'm': Complete (maximum) pairwise linkage (default) - *method* == 'c': Centroid linkage - *method* == 'a': Average pairwise linkage
- *distancematrix*: The distance matrix between the items. There are three ways in which you can pass a distance matrix: 1. a 2D Numerical Python array (in which only the left-lower part of the array will be accessed); 2. a 1D Numerical Python array containing the distances consecutively; 3. a list of rows containing the lower-triangular part of the distance matrix.

Examples are:

```
>>> from numpy import array
>>> # option 1:
>>> distance = array([[0.0, 1.1, 2.3],
...                  [1.1, 0.0, 4.5],
...                  [2.3, 4.5, 0.0]])
>>> # option 2:
>>> distance = array([1.1, 2.3, 4.5])
>>> # option 3:
>>> distance = [array([]),
...             array([1.1]),
...             array([2.3, 4.5])]
```

These three correspond to the same distance matrix.

PLEASE NOTE: As the `treecluster` routine may shuffle the values in the distance matrix as part of the clustering algorithm, be sure to save this array in a different variable before calling `treecluster` if you need it later.

Either `data` or `distancematrix` should be `None`. If `distancematrix` is `None`, the hierarchical clustering solution is calculated from the values stored in the argument `data`. If `data` is `None`, the hierarchical clustering solution is instead calculated from the distance matrix. Pairwise centroid-linkage clustering can be performed only from the data values and not from the distance matrix. Pairwise single-, maximum-, and average-linkage clustering can be calculated from the data values or from the distance matrix.

Return value: `treecluster` returns a `Tree` object describing the hierarchical clustering result. See the description of the `Tree` class for more information.

6.3.5 `pyms.Experiment`

Models a GC-MS experiment, represented by a list of signal peaks.

Classes:

<code>Experiment(expr_code, peak_list)</code>	Models an experiment.
---	-----------------------

Functions:

<code>load_expr(file_name)</code>	Loads an experiment saved with <code>pyms.Experiment.Experiment.dump()</code> .
<code>read_expr_list(file_name)</code>	Reads the set of experiment files and returns a list of <code>pyms.Experiment.Experiment</code> objects.

class `Experiment` (*expr_code*, *peak_list*)

Bases: `pymsBaseClass`

Models an experiment.

Parameters

- **expr_code** (`str`) – A unique identifier for the experiment.
- **peak_list** (`Sequence[Peak]`)

Author Vladimir Likic, Andrew Isaac, Dominic Davis-Foster (type assertions, properties and pathlib support)

Methods:

<code>__copy__()</code>	Returns a new <code>Experiment</code> object containing a copy of the data in this object.
<code>__deepcopy__([memodict])</code>	Returns a new <code>Experiment</code> object containing a copy of the data in this object.
<code>__eq__(other)</code>	Return whether this <code>Experiment</code> object is equal to another object.
<code>__len__()</code>	Returns the number of peaks in the <code>Experiment</code> .
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>select_rt_range(rt_range)</code>	Discards all peaks which have the retention time outside the specified range.

Attributes:

<code>expr_code</code>	Returns the <code>expr_code</code> of the experiment.
<code>peak_list</code>	Returns the peak list.

`__copy__()`

Returns a new Experiment object containing a copy of the data in this object.

Return type `Experiment`

`__deepcopy__(memodict={})`

Returns a new Experiment object containing a copy of the data in this object.

Return type `Experiment`

`__eq__(other)`

Return whether this Experiment object is equal to another object.

Parameters `other` – The other object to test equality with.

Return type `bool`

`__len__()`

Returns the number of peaks in the Experiment.

Return type `int`

`dump(file_name, protocol=3)`

Dumps an object to a file through `pickle.dump()`.

Parameters

- **`file_name`** (`Union[str, Path, PathLike]`) – Filename to save the dump as.
- **`protocol`** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

`property expr_code`

Returns the `expr_code` of the experiment.

Return type `str`

`property peak_list`

Returns the peak list.

Return type `List[Peak]`

`sele_rt_range(rt_range)`

Discards all peaks which have the retention time outside the specified range.

Parameters **`rt_range`** (`Sequence[str]`) – Min, max retention time given as a sequence `[rt_min, rt_max]`.

`load_expr(file_name)`

Loads an experiment saved with `pymz.Experiment.Experiment.dump()`.

Parameters `file_name` (`Union[str, Path, PathLike]`) – Experiment file name.

Return type `Experiment`

Returns The loaded experiment.

Author Vladimir Likic, Andrew Isaac, Dominic Davis-Foster (type assertions and pathlib support)

read_expr_list (`file_name`)

Reads the set of experiment files and returns a list of `pym.s.Experiment.Experiment` objects.

Parameters `file_name` (`Union[str, Path, PathLike]`) – The name of the file which lists experiment dump file names, one file per line.

Return type `List[Experiment]`

Returns A list of Experiment instances.

Author Vladimir Likic

6.3.6 pym.s.Gapfill

Table of Contents

- `pym.s.Gapfill`
 - `pym.s.Gapfill.Class`
 - `pym.s.Gapfill.Function`

Gap Filling Routines.

`pym.s.Gapfill.Class`

Provides a class for handling Missing Peaks in an output file (i.e. `area.csv`).

Classes:

<code>MissingPeak</code> (<code>common_ion</code> , <code>qual_ion_2</code>)	<code>qual_ion_1</code> ,	Class to encapsulate a peak object identified as missing in the output area matrix fom PyMassSpec.
<code>Sample</code> (<code>sample_name</code> , <code>matrix_position</code>)		A collection of MissingPeak objects.

class MissingPeak (`common_ion`, `qual_ion_1`, `qual_ion_2`, `rt=0.0`)

Bases: `object`

Class to encapsulate a peak object identified as missing in the output area matrix fom PyMassSpec.

Parameters

- **common_ion** (`int`) – Common ion for the peak across samples in an experiment.
- **qual_ion_1** (`int`) – The top (most abundant) ion for the peak object
- **qual_ion_2** (`int`) – The second most abundant ion for the peak object
- **rt** (`float`) – Retention time of the peak. Default 0.0.

Authors Jairus Bowne, Sean O’Callaghan, Dominic Davis-Foster

Attributes:

<code>common_ion</code>	Returns the common ion for the peak object across an experiment.
<code>common_ion_area</code>	The area of the common ion
<code>exact_rt</code>	The retention time of the apex of the peak
<code>qual_ion1</code>	Returns the top (most abundant) ion for the peak object.
<code>qual_ion2</code>	Returns the second most abundant ion for the peak object.
<code>rt</code>	Returns the retention time of the peak.

property common_ion

Returns the common ion for the peak object across an experiment.

Return type `int`

Returns Common ion for the peak

Author Jairus Bowne

common_ion_area

Type: `Optional[float]`

The area of the common ion

exact_rt

Type: `Optional[float]`

The retention time of the apex of the peak

property qual_ion1

Returns the top (most abundant) ion for the peak object.

Return type `int`

Returns Most abundant ion

Author Jairus Bowne

property qual_ion2

Returns the second most abundant ion for the peak object.

Return type `int`

Returns Second most abundant ion

Author Jairus Bowne

property rt

Returns the retention time of the peak.

Return type `float`

class Sample (*sample_name*, *matrix_position*)

Bases: `object`

A collection of MissingPeak objects.

Parameters

- **sample_name** (`str`) – the experiment code/name.
- **matrix_position** (`int`) – position along x-axis where sample is located.

Authors Sean O’Callaghan, Dominic Davis-Foster (properties)

Methods:

<code>add_missing_peak(missing_peak)</code>	Add a new MissingPeak object to the Sample.
<code>get_mp_rt_exact_rt_dict()</code>	Returns a dictionary containing <code>average_rt</code> : <code>exact_rt</code> pairs.

Attributes:

<code>missing_peaks</code>	Returns a list of the MissingPeak objects in the Sample object.
<code>name</code>	Returns name of the sample.
<code>rt_areas</code>	Returns a dictionary containing <code>rt</code> : <code>area</code> pairs.

add_missing_peak (*missing_peak*)

Add a new MissingPeak object to the Sample.

Parameters **missing_peak** (*MissingPeak*) – The missing peak object to be added.

get_mp_rt_exact_rt_dict ()

Returns a dictionary containing `average_rt` : `exact_rt` pairs.

Return type `Dict[float, Optional[float]]`

property missing_peaks

Returns a list of the MissingPeak objects in the Sample object.

Return type `List[MissingPeak]`

property name

Returns name of the sample.

Return type `str`

property rt_areas

Returns a dictionary containing `rt` : `area` pairs.

Return type `Dict[float, Optional[float]]`

pyms.Gapfill.Function

Functions to fill missing peak objects.

Classes:

<code>MissingPeakFiletype(value)</code>	Flag to indicate the filetype for <code>pyms.Gapfill.Function.missing_peak_finder()</code> .
---	--

Functions:

<code>file2dataframe(file_name)</code>	Convert a .csv file to a pandas DataFrame.
<code>missing_peak_finder(sample, file_name[, ...])</code>	Integrates raw data around missing peak locations to fill NAs in the data matrix.
<code>mp_finder(input_matrix)</code>	Finds the 'NA's in the transformed <code>area_ci.csv</code> file and makes <code>pyms.Gapfill.Class.Sample</code> objects with them
<code>write_filled_csv(sample_list, area_file, ...)</code>	Creates a new <code>area_ci.csv</code> file, replacing NAs with values from the <code>sample_list</code> objects where possible.
<code>write_filled_rt_csv(sample_list, rt_file, ...)</code>	Creates a new <code>rt.csv</code> file, replacing 'NA's with values from the <code>sample_list</code> objects where possible.

enum MissingPeakFiletype (value)

Bases: `enum_tools.custom_enums.IntEnum`

Flag to indicate the filetype for `pyms.Gapfill.Function.missing_peak_finder()`.

New in version 2.3.0.

Member Type int

Valid values are as follows:

MZML = `<MissingPeakFiletype.MZML: 1>`

NETCDF = `<MissingPeakFiletype.NETCDF: 2>`

file2dataframe (file_name)

Convert a .csv file to a pandas DataFrame.

Parameters `file_name` (`Union[str, Path, PathLike]`) – CSV file to read.

Authors Jairus Bowne, Sean O’Callaghan, Dominic Davis-Foster (pathlib support)

New in version 2.3.0.

Return type `DataFrame`

missing_peak_finder (sample, file_name, points=3, null_ions=None, crop_ions=None, threshold=1000, rt_window=1, filetype=<MissingPeakFiletype.MZML: 1>)

Integrates raw data around missing peak locations to fill NAs in the data matrix.

Parameters

- **sample** (`Sample`) – The sample object containing missing peaks
- **file_name** (`str`) – Name of the raw data file

- **points** (*int*) – Peak finding - Peak if maxima over ‘points’ number of scans. Default 3.
- **null_ions** (*Optional[List]*) – Ions to be deleted in the matrix. Default [73, 147].
- **crop_ions** (*Optional[List]*) – Range of Ions to be considered. Default [50, 540].
- **threshold** (*int*) – Minimum intensity of IonChromatogram allowable to fill. Default 1000.
- **rt_window** (*float*) – Window in seconds around average RT to look for. Default 1.
- **filetype** (*MissingPeakFiletype*) – Default <MissingPeakFiletype.MZML: 1>.

Author Sean O’Callaghan

mp_finder (*input_matrix*)

Finds the 'NA's in the transformed `area_ci.csv` file and makes `pymz.Gapfill.Class.Sample` objects with them

Parameters **input_matrix** (*List*) – Data matrix derived from the `area_ci.csv` file.

Return type *List[Sample]*

Authors Jairus Bowne, Sean O’Callaghan

write_filled_csv (*sample_list, area_file, filled_area_file*)

Creates a new `area_ci.csv` file, replacing NAs with values from the `sample_list` objects where possible.

Parameters

- **sample_list** (*List[Sample]*)
- **area_file** (*Union[str, Path, PathLike]*) – The file '`area_ci.csv`' from PyMassSpec output.
- **filled_area_file** (*Union[str, Path, PathLike]*) – the new output file which has 'NA's values replaced.

Authors Jairus Bowne, Sean O’Callaghan, Dominic Davis-Foster

write_filled_rt_csv (*sample_list, rt_file, filled_rt_file*)

Creates a new `rt.csv` file, replacing 'NA's with values from the `sample_list` objects where possible.

Parameters

- **sample_list** (*List[Sample]*) – A list of samples.
- **rt_file** (*Union[str, Path, PathLike]*) – the file `rt.csv` from PyMassSpec output.
- **filled_rt_file** (*Union[str, Path, PathLike]*) – the new output file which has NA values replaced.

Authors Jairus Bowne, Sean O’Callaghan, Dominic Davis-Foster

6.3.7 `pyms.GCMS`

Table of Contents

- `pyms.GCMS`
 - `pyms.GCMS.Function`
 - `pyms.GCMS.IO`
 - * `pyms.GCMS.IO.ANDI`
 - * `pyms.GCMS.IO.JCAMP`
 - * `pyms.GCMS.IO.MZML`

Module to handle raw data.

`pyms.GCMS.Class`

Class to model GC-MS data.

Classes:

<code>GCMS_data</code> (<code>time_list</code> , <code>scan_list</code>)	Generic object for GC-MS data.
--	--------------------------------

Data:

<code>IntStr</code>	Invariant <code>TypeVar</code> constrained to <code>int</code> and <code>str</code> .
---------------------	---

class `GCMS_data`(`time_list`, `scan_list`)

Bases: `pymsBaseClass`, `TimeListMixin`, `MaxMinMassMixin`, `GetIndexTimeMixin`

Generic object for GC-MS data.

Contains the raw data as a list of scans and a list of times.

Parameters

- **`time_list`** (`Sequence[float]`) – Scan retention times.
- **`scan_list`** (`Sequence[Scan]`)

Authors Qiao Wang, Andrew Isaac, Vladimir Likic, Dominic Davis-Foster (type assertions and properties)

Methods:

<code>__eq__</code> (<code>other</code>)	Return whether this <code>GCMS_data</code> object is equal to another object.
<code>__len__</code> ()	Returns the length of the data object, defined as the number of scans.
<code>__repr__</code> ()	Return a string representation of the <code>GCMS_data</code> .
<code>__str__</code> ()	Return <code>str(self)</code> .
<code>dump</code> (<code>file_name</code> [, <code>protocol</code>])	Dumps an object to a file through <code>pickle.dump()</code> .

continues on next page

Table 28 – continued from previous page

<code>get_index_at_time(time)</code>	Returns the nearest index corresponding to the given time.
<code>get_time_at_index(ix)</code>	Returns time at given index.
<code>info([print_scan_n])</code>	Prints some information about the data.
<code>trim([begin, end])</code>	Trims data in the time domain.
<code>write(file_root)</code>	Writes the entire raw data to two CSV files:
<code>write_intensities_stream(file_name)</code>	Loop over all scans and, for each scan, write the intensities to the given file, one intensity per line.

Attributes:

<code>max_mass</code>	Returns the maximum m/z value in the spectrum.
<code>max_rt</code>	Returns the maximum retention time for the data in seconds.
<code>min_mass</code>	Returns the minimum m/z value in the spectrum.
<code>min_rt</code>	Returns the minimum retention time for the data in seconds.
<code>scan_list</code>	Return a list of the scan objects.
<code>tic</code>	Returns the total ion chromatogram.
<code>time_list</code>	Return a copy of the time list.
<code>time_step</code>	Returns the time step of the data.
<code>time_step_std</code>	Returns the standard deviation of the time step of the data.

`__eq__ (other)`

Return whether this GCMS_data object is equal to another object.

Parameters `other` – The other object to test equality with.

Return type `bool`

`__len__ ()`

Returns the length of the data object, defined as the number of scans.

Author Vladimir Likic

Return type `int`

`__repr__ ()`

Return a string representation of the `GCMS_data`.

Return type `str`

`__str__ ()`

Return `str(self)`.

Return type `str`

`dump (file_name, protocol=3)`

Dumps an object to a file through `pickle.dump()`.

Parameters

- `file_name` (`Union[str, Path, PathLike]`) – Filename to save the dump as.

- **protocol** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

get_index_at_time (*time*)

Returns the nearest index corresponding to the given time.

Parameters **time** (`float`) – Time in seconds

Return type `int`

Returns Nearest index corresponding to given time

Authors Lewis Lee, Tim Erwin, Vladimir Likic

Changed in version 2.3.0: Now returns -1 if no index is found.

get_time_at_index (*ix*)

Returns time at given index.

Parameters **ix** (`int`)

Authors Lewis Lee, Vladimir Likic

Return type `float`

info (*print_scan_n=False*)

Prints some information about the data.

Parameters **print_scan_n** (`bool`) – If set to `True` will print the number of m/z values in each scan.
Default `False`.

Author Vladimir Likic

property max_mass

Returns the maximum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

property max_rt

Returns the maximum retention time for the data in seconds.

Return type `float`

property min_mass

Returns the minimum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

property min_rt

Returns the minimum retention time for the data in seconds.

Return type `float`

property scan_list

Return a list of the scan objects.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List[Scan]`

property tic

Returns the total ion chromatogram.

Author Andrew Isaac

Return type `IonChromatogram`

property time_list

Return a copy of the time list.

Return type `List[float]`

property time_step

Returns the time step of the data.

Return type `float`

property time_step_std

Returns the standard deviation of the time step of the data.

Return type `float`

trim (*begin=None, end=None*)

Trims data in the time domain.

The arguments *begin* and *end* can be either integers (in which case they are taken as the first/last scan number for trimming) or strings in which case they are treated as time strings and converted to scan numbers.

At present both *begin* and *end* must be of the same type, either both scan numbers or time strings.

At least one of *begin* and *end* is required.

Parameters

- **begin** (`Optional[~IntStr]`) – The start time or scan number. Default `None`.
- **end** (`Optional[~IntStr]`) – The end time or scan number. Default `None`.

Author Vladimir Likic

write (*file_root*)

Writes the entire raw data to two CSV files:

- `<file_root>.I.csv`, containing the intensities; and
- `<file_root>.mz.csv`, containing the corresponding *m/z* values.

In general these are not two-dimensional matrices, because different scans may have different numbers of *m/z* values recorded.

Parameters **file_root** (`Union[str, Path, PathLike]`) – The root for the output file names

Authors Vladimir Likic, Dominic Davis-Foster (pathlib support)

write_intensities_stream (*file_name*)

Loop over all scans and, for each scan, write the intensities to the given file, one intensity per line.

Intensities from different scans are joined without any delimiters.

Parameters **file_name** (`Union[str, Path, PathLike]`) – Output file name.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib support)

IntStr = `TypeVar(IntStr, int, str)`

Type: `TypeVar`

Invariant `TypeVar` constrained to `int` and `str`.

pyms.GCMS.Function

Provides conversion and information functions for GC-MS data objects.

Functions:

<i>diff</i> (data1, data2)	Compares two GCMS_data objects.
<i>ic_window_points</i> (ic, window_sele[, half_window])	Converts the window selection parameter into points based on the time step in an ion chromatogram.

diff (*data1*, *data2*)

Compares two GCMS_data objects.

Parameters

- **data1** (*GCMS_data*) – GCMS data set 1
- **data2** (*GCMS_data*) – GCMS data set 2

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

ic_window_points (*ic*, *window_sele*, *half_window=False*)

Converts the window selection parameter into points based on the time step in an ion chromatogram.

Parameters

- **ic** (*IonChromatogram*) – ion chromatogram object relevant for the conversion
- **window_sele** (`Union[int, str]`) – The window selection parameter. This can be an integer or time string. If an integer, taken as the number of points. If a string, must of the form '`<NUMBER>s`' or '`<NUMBER>m`', specifying a time in seconds or minutes, respectively
- **half_window** (`bool`) – Specifies whether to return half-window. Default `False`.

Author Vladimir Likic

Return type `int`

pyms.GCMS.IO

Input/output functions for GC-MS data files.

pyms.GCMS.IO.ANDI

Functions for reading ANDI-MS data files.

Functions:

<code>ANDI_reader(file_name)</code>	A reader for ANDI-MS NetCDF files.
-------------------------------------	------------------------------------

ANDI_reader (*file_name*)

A reader for ANDI-MS NetCDF files.

Parameters `file_name` (`Union[str, Path, PathLike]`) – The path of the ANDI-MS file

Return type `GCMS_data`

Returns GC-MS data object

Authors Qiao Wang, Andrew Isaac, Vladimir Likic, Dominic Davis-Foster

pyms.GCMS.IO.JCAMP

Functions for I/O of data in JCAMP-DX format.

Functions:

<code>JCAMP_reader(file_name)</code>	Generic reader for JCAMP DX files.
--------------------------------------	------------------------------------

JCAMP_reader (*file_name*)

Generic reader for JCAMP DX files.

Parameters `file_name` (`Union[str, Path]`) – Path of the file to read

Return type `GCMS_data`

Returns GC-MS data object

Authors Qiao Wang, Andrew Isaac, Vladimir Likic, David Kainer, Dominic Davis-Foster (pathlib support)

pyms.GCMS.IO.MZML

Functions for reading mzML format data files.

Functions:

<code>mzML_reader(file_name)</code>	A reader for mzML files.
-------------------------------------	--------------------------

mzML_reader (*file_name*)

A reader for mzML files.

Parameters `file_name` (`Union[str, Path, PathLike]`) – The name of the mzML file.

Return type `GCMS_data`

Returns GC-MS data object.

Authors Sean O’Callaghan, Dominic Davis-Foster (pathlib support)

6.3.8 `pyms.IntensityMatrix`

Class to model Intensity Matrix.

Classes:

<code>AsciiFiletypes(value)</code>	Enumeration of supported ASCII filetypes for <code>export_ascii()</code> .
<code>BaseIntensityMatrix(time_list, mass_list, ...)</code>	Base class for intensity matrices of binned raw data.
<code>IntensityMatrix(time_list, mass_list, ...)</code>	Intensity matrix of binned raw data.

Functions:

<code>build_intensity_matrix(data[, bin_interval, ...])</code>	Sets the full intensity matrix with flexible bins.
<code>build_intensity_matrix_i(data[, bin_left, ...])</code>	Sets the full intensity matrix with integer bins.
<code>import_leco_csv(file_name)</code>	Imports data in LECO CSV format.

enum `AsciiFiletypes` (*value*)

Bases: `enum_tools.custom_enums.IntEnum`

Enumeration of supported ASCII filetypes for `export_ascii()`.

New in version 2.3.0.

Member Type `int`

Valid values are as follows:

ASCII_DAT = `<AsciiFiletypes.ASCII_DAT: 1>`

Tab-delimited ASCII file

ASCII_CSV = `<AsciiFiletypes.ASCII_CSV: 0>`

Comma-separated values file

class `BaseIntensityMatrix` (*time_list, mass_list, intensity_array*)

Bases: `pymsBaseClass`, `TimeListMixin`, `MassListMixin`, `IntensityArrayMixin`, `GetIndexTimeMixin`

Base class for intensity matrices of binned raw data.

Parameters

- **time_list** (`Sequence[float]`) – Retention time values
- **mass_list** (`Sequence[float]`) – Binned mass values

- **intensity_array** (`Union[Sequence[Sequence[float]], ndarray]`) – List of lists of binned intensity values per scan

Authors Andrew Isaac, Dominic Davis-Foster (type assertions and properties)

Methods:

<code>__eq__(other)</code>	Return whether this intensity matrix object is equal to another object.
<code>__len__()</code>	Returns the number of scans in the intensity matrix.
<code>crop_mass(mass_min, mass_max)</code>	Crops mass spectrum.
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>get_ic_at_index(ix)</code>	Returns the ion chromatogram at the specified index.
<code>get_index_at_time(time)</code>	Returns the nearest index corresponding to the given time.
<code>get_index_of_mass(mass)</code>	Returns the index of the nearest binned mass to the given mass.
<code>get_mass_at_index(ix)</code>	Returns binned mass at index.
<code>get_ms_at_index(ix)</code>	Returns a mass spectrum for a given scan index.
<code>get_scan_at_index(ix)</code>	Returns the spectral intensities for scan index.
<code>get_time_at_index(ix)</code>	Returns time at given index.
<code>iter_ic_indices()</code>	Iterate over column indices.
<code>iter_ms_indices()</code>	Iterates over row indices.
<code>null_mass(mass)</code>	Ignore given (closest) mass in spectra.
<code>reduce_mass_spectra([n_intensities])</code>	Reduces the mass spectra by retaining the top <i>n_intensities</i> , discarding all other intensities.
<code>set_ic_at_index(ix, ic)</code>	Sets the intensity of the mass at index <i>ix</i> in each scan to a new value.

Attributes:

<code>intensity_array</code>	Returns a copy of the intensity array.
<code>intensity_array_list</code>	Returns a copy of the intensity array as a list of lists of floats.
<code>intensity_matrix</code>	Returns a copy of the intensity matrix.
<code>mass_list</code>	Returns a list of the masses.
<code>matrix_list</code>	Returns the intensity matrix as a list of lists of floats.
<code>max_mass</code>	Returns the maximum <i>m/z</i> value in the spectrum.
<code>min_mass</code>	Returns the minimum <i>m/z</i> value in the spectrum.
<code>size</code>	Gets the size of intensity matrix.
<code>time_list</code>	Returns a copy of the time list.

`__eq__(other)`
Return whether this intensity matrix object is equal to another object.

Parameters *other* – The other object to test equality with.

Return type `bool`

`__len__()`
Returns the number of scans in the intensity matrix.

Return type `int`

crop_mass (*mass_min*, *mass_max*)
Crops mass spectrum.

Parameters

- **mass_min** (`float`) – Minimum mass value
- **mass_max** (`float`) – Maximum mass value

Author Andrew Isaac

dump (*file_name*, *protocol*=3)
Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – Filename to save the dump as.
- **protocol** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

get_ic_at_index (*ix*)
Returns the ion chromatogram at the specified index.

Parameters **ix** (`int`) – Index of an ion chromatogram in the intensity data matrix.

Return type `IonChromatogram`

Returns Ion chromatogram at given index.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

get_index_at_time (*time*)
Returns the nearest index corresponding to the given time.

Parameters **time** (`float`) – Time in seconds

Return type `int`

Returns Nearest index corresponding to given time

Authors Lewis Lee, Tim Erwin, Vladimir Likic

Changed in version 2.3.0: Now returns -1 if no index is found.

get_index_of_mass (*mass*)
Returns the index of the nearest binned mass to the given mass.

Parameters **mass** (`float`) – Mass to lookup in list of masses

Author Andrew Isaac

Return type `int`

get_mass_at_index (*ix*)
Returns binned mass at index.

Parameters **ix** (`int`) – Index of binned mass

Return type `float`

Returns Binned mass

Author Andrew Isaac

get_ms_at_index (*ix*)

Returns a mass spectrum for a given scan index.

Parameters *ix* (*int*) – The index of the scan.

Author Andrew Isaac

Return type *MassSpectrum*

get_scan_at_index (*ix*)

Returns the spectral intensities for scan index.

Parameters *ix* (*int*) – The index of the scan

Return type *List[float]*

Returns Intensity values of scan spectra

Author Andrew Isaac

get_time_at_index (*ix*)

Returns time at given index.

Parameters *ix* (*int*)

Authors Lewis Lee, Vladimir Likic

Return type *float*

property intensity_array

Returns a copy of the intensity array.

Return type *ndarray*

Returns Matrix of intensity values.

Authors Andrew Isaac, Lewis Lee

property intensity_array_list

Returns a copy of the intensity array as a list of lists of floats.

Return type *List[List[float]]*

Returns Matrix of intensity values.

Author Andrew Isaac

property intensity_matrix

Returns a copy of the intensity matrix.

Return type *ndarray*

Returns Matrix of intensity values.

Author Andrew Isaac

iter_ic_indices()

Iterate over column indices.

Return type `Iterator[int]`

iter_ms_indices()

Iterates over row indices.

Return type `Iterator[int]`

property mass_list

Returns a list of the masses.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List[float]`

property matrix_list

Returns the intensity matrix as a list of lists of floats.

Return type `ndarray`

Returns Matrix of intensity values

Author Andrew Isaac

property max_mass

Returns the maximum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

property min_mass

Returns the minimum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

null_mass(*mass*)

Ignore given (closest) mass in spectra.

Parameters **mass** (`float`) – Mass value to remove

Author Andrew Isaac

reduce_mass_spectra(*n_intensities*=5)

Reduces the mass spectra by retaining the top $n_intensities$, discarding all other intensities.

Parameters **n_intensities** (`int`) – The number of top intensities to keep. Default 5.

Author Vladimir Likic

set_ic_at_index(*ix*, *ic*)

Sets the intensity of the mass at index `ix` in each scan to a new value.

Parameters

- **ix** (*int*) – Index of an ion chromatogram in the intensity data matrix to be set
- **ic** (*IonChromatogram*) – Ion chromatogram that will be copied at position *ix* in the data matrix

The length of the ion chromatogram must match the appropriate dimension of the intensity matrix.

Author Vladimir Likic

property size

Gets the size of intensity matrix.

Return type *Tuple*[*int*, *int*]

Returns Number of rows and cols

Authors Qiao Wang, Andrew Isaac, Luke Hodgkinson, Vladimir Likic

property time_list

Returns a copy of the time list.

Return type *List*[*float*]

Returns List of retention times

Authors Andrew Isaac, Lewis Lee, Vladimir Likic

class IntensityMatrix (*time_list*, *mass_list*, *intensity_array*)

Bases: *BaseIntensityMatrix*

Intensity matrix of binned raw data.

Parameters

- **time_list** (*Sequence*[*float*]) – Retention time values
- **mass_list** (*Sequence*[*float*]) – Binned mass values
- **intensity_array** (*Union*[*Sequence*[*Sequence*[*float*]], *ndarray*]) – List of lists of binned intensity values per scan

Authors Andrew Isaac, Dominic Davis-Foster (type assertions and properties)

Methods:

<code>__eq__(other)</code>	Return whether this intensity matrix object is equal to another object.
<code>__len__()</code>	Returns the number of scans in the intensity matrix.
<code>crop_mass(mass_min, mass_max)</code>	Crops mass spectrum.
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>export_ascii(root_name[, fmt])</code>	Exports the intensity matrix, retention time vector, and m/z vector to the ascii format.
<code>export_leco_csv(file_name)</code>	Exports data in LECO CSV format.
<code>get_ic_at_index(ix)</code>	Returns the ion chromatogram at the specified index.
<code>get_ic_at_mass([mass])</code>	Returns the ion chromatogram for the nearest binned mass to the specified mass.

continues on next page

Table 38 – continued from previous page

<code>get_index_at_time(time)</code>	Returns the nearest index corresponding to the given time.
<code>get_index_of_mass(mass)</code>	Returns the index of the nearest binned mass to the given mass.
<code>get_mass_at_index(ix)</code>	Returns binned mass at index.
<code>get_ms_at_index(ix)</code>	Returns a mass spectrum for a given scan index.
<code>get_scan_at_index(ix)</code>	Returns the spectral intensities for scan index.
<code>get_time_at_index(ix)</code>	Returns time at given index.
<code>iter_ic_indices()</code>	Iterate over local column indices.
<code>iter_ms_indices()</code>	Iterates over the local row indices.
<code>null_mass(mass)</code>	Ignore given (closest) mass in spectra.
<code>reduce_mass_spectra([n_intensities])</code>	Reduces the mass spectra by retaining the top <i>n_intensities</i> , discarding all other intensities.
<code>set_ic_at_index(ix, ic)</code>	Sets the intensity of the mass at index <i>ix</i> in each scan to a new value.

Attributes:

<code>bpc</code>	Constructs a Base Peak Chromatogram from the data.
<code>intensity_array</code>	Returns a copy of the intensity array.
<code>intensity_array_list</code>	Returns a copy of the intensity array as a list of lists of floats.
<code>intensity_matrix</code>	Returns a copy of the intensity matrix.
<code>local_size</code>	Gets the local size of intensity matrix.
<code>mass_list</code>	Returns a list of the masses.
<code>matrix_list</code>	Returns the intensity matrix as a list of lists of floats.
<code>max_mass</code>	Returns the maximum m/z value in the spectrum.
<code>min_mass</code>	Returns the minimum m/z value in the spectrum.
<code>size</code>	Gets the size of intensity matrix.
<code>tic</code>	Returns the TIC of the intensity matrix.
<code>time_list</code>	Returns a copy of the time list.

`__eq__ (other)`

Return whether this intensity matrix object is equal to another object.

Parameters `other` – The other object to test equality with.

Return type `bool`

`__len__ ()`

Returns the number of scans in the intensity matrix.

Return type `int`

property `bpc`

Constructs a Base Peak Chromatogram from the data.

This represents the most intense ion for each scan.

Authors Dominic Davis-Foster

New in version 2.3.0.

Return type *IonChromatogram*

crop_mass (*mass_min*, *mass_max*)

Crops mass spectrum.

Parameters

- **mass_min** (*float*) – Minimum mass value
- **mass_max** (*float*) – Maximum mass value

Author Andrew Isaac

dump (*file_name*, *protocol*=3)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (*Union[str, Path, PathLike]*) – Filename to save the dump as.
- **protocol** (*int*) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

export_ascii (*root_name*, *fmt*=<AsciiFiletypes.ASCII_DAT: 1>)

Exports the intensity matrix, retention time vector, and m/z vector to the ascii format.

By default, `export_ascii("NAME")` will create `NAME.im.dat`, `NAME.rt.dat`, and `NAME.mz.dat` where these are the intensity matrix, retention time vector, and m/z vector in tab delimited format.

If `format == <AsciiFiletypes.ASCII_CSV>`, the files will be in the CSV format, named `NAME.im.csv`, `NAME.rt.csv`, and `NAME.mz.csv`.

Parameters

- **root_name** (*Union[str, Path, PathLike]*) – Root name for the output files
- **fmt** (*AsciiFiletypes*) – Format of the output file, either `<AsciiFiletypes.ASCII_DAT>` or `<AsciiFiletypes.ASCII_CSV>`. Default `<AsciiFiletypes.ASCII_DAT: 1>`.

Authors Milica Ng, Andrew Isaac, Vladimir Likic, Dominic Davis-Foster (pathlib support)

export_leco_csv (*file_name*)

Exports data in LECO CSV format.

Parameters **file_name** (*Union[str, Path, PathLike]*) – The name of the output file.

Authors Andrew Isaac, Vladimir Likic, Dominic Davis-Foster (pathlib support)

get_ic_at_index (*ix*)

Returns the ion chromatogram at the specified index.

Parameters **ix** (*int*) – Index of an ion chromatogram in the intensity data matrix.

Return type *IonChromatogram*

Returns Ion chromatogram at given index.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

get_ic_at_mass (*mass=None*)

Returns the ion chromatogram for the nearest binned mass to the specified mass.

If no mass value is given, the function returns the total ion chromatogram.

Parameters **mass** (*Optional[float]*) – Mass value of an ion chromatogram. Default *None*.

Return type *IonChromatogram*

Returns Ion chromatogram for given mass

Authors Andrew Isaac, Vladimir Likic

get_index_at_time (*time*)

Returns the nearest index corresponding to the given time.

Parameters **time** (*float*) – Time in seconds

Return type *int*

Returns Nearest index corresponding to given time

Authors Lewis Lee, Tim Erwin, Vladimir Likic

Changed in version 2.3.0: Now returns -1 if no index is found.

get_index_of_mass (*mass*)

Returns the index of the nearest binned mass to the given mass.

Parameters **mass** (*float*) – Mass to lookup in list of masses

Author Andrew Isaac

Return type *int*

get_mass_at_index (*ix*)

Returns binned mass at index.

Parameters **ix** (*int*) – Index of binned mass

Return type *float*

Returns Binned mass

Author Andrew Isaac

get_ms_at_index (*ix*)

Returns a mass spectrum for a given scan index.

Parameters **ix** (*int*) – The index of the scan.

Author Andrew Isaac

Return type *MassSpectrum*

get_scan_at_index (*ix*)

Returns the spectral intensities for scan index.

Parameters **ix** (*int*) – The index of the scan

Return type *List[float]*

Returns Intensity values of scan spectra

Author Andrew Isaac

get_time_at_index (*ix*)
Returns time at given index.

Parameters *ix* (*int*)

Authors Lewis Lee, Vladimir Likic

Return type *float*

property intensity_array
Returns a copy of the intensity array.

Return type *ndarray*

Returns Matrix of intensity values.

Authors Andrew Isaac, Lewis Lee

property intensity_array_list
Returns a copy of the intensity array as a list of lists of floats.

Return type *List[List[float]]*

Returns Matrix of intensity values.

Author Andrew Isaac

property intensity_matrix
Returns a copy of the intensity matrix.

Return type *ndarray*

Returns Matrix of intensity values.

Author Andrew Isaac

iter_ic_indices ()
Iterate over local column indices.

Author Luke Hodkinson

Return type *Iterator[int]*

iter_ms_indices ()
Iterates over the local row indices.

Author Luke Hodkinson

Return type *Iterator[int]*

property local_size
Gets the local size of intensity matrix.

Returns Number of rows and cols

Return type *int*

Author Luke Hodkinson

property mass_list

Returns a list of the masses.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List[float]`

property matrix_list

Returns the intensity matrix as a list of lists of floats.

Return type `ndarray`

Returns Matrix of intensity values

Author Andrew Isaac

property max_mass

Returns the maximum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

property min_mass

Returns the minimum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

null_mass (*mass*)

Ignore given (closest) mass in spectra.

Parameters **mass** (`float`) – Mass value to remove

Author Andrew Isaac

reduce_mass_spectra (*n_intensities=5*)

Reduces the mass spectra by retaining the top *n_intensities*, discarding all other intensities.

Parameters **n_intensities** (`int`) – The number of top intensities to keep. Default 5.

Author Vladimir Likic

set_ic_at_index (*ix, ic*)

Sets the intensity of the mass at index *ix* in each scan to a new value.

Parameters

- **ix** (`int`) – Index of an ion chromatogram in the intensity data matrix to be set
- **ic** (*IonChromatogram*) – Ion chromatogram that will be copied at position *ix* in the data matrix

The length of the ion chromatogram must match the appropriate dimension of the intensity matrix.

Author Vladimir Likic

property size

Gets the size of intensity matrix.

Return type `Tuple[int, int]`

Returns Number of rows and cols

Authors Qiao Wang, Andrew Isaac, Luke Hodgkinson, Vladimir Likic

property tic

Returns the TIC of the intensity matrix.

New in version 2.3.0.

Return type `IonChromatogram`

property time_list

Returns a copy of the time list.

Return type `List[float]`

Returns List of retention times

Authors Andrew Isaac, Lewis Lee, Vladimir Likic

build_intensity_matrix (*data*, *bin_interval*=1, *bin_left*=0.5, *bin_right*=0.5, *min_mass*=None)
Sets the full intensity matrix with flexible bins.

The first bin is centered around *min_mass*, and subsequent bins are offset by *bin_interval*.

Parameters

- **data** (*GCMS_data*) – Raw GCMS data
- **bin_interval** (*float*) – interval between bin centres. Default 1.
- **bin_left** (*float*) – left bin boundary offset. Default 0.5.
- **bin_right** (*float*) – right bin boundary offset. Default 0.5.
- **min_mass** (*Optional[float]*) – Minimum mass to bin (default minimum mass from data). Default *None*.

Return type `IntensityMatrix`

Returns Binned IntensityMatrix object

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

build_intensity_matrix_i (*data*, *bin_left*=0.3, *bin_right*=0.7)
Sets the full intensity matrix with integer bins.

Parameters

- **data** (*GCMS_data*) – Raw GCMS data
- **bin_left** (*float*) – left bin boundary offset. Default 0.3.
- **bin_right** (*float*) – right bin boundary offset. Default 0.7.

Return type `IntensityMatrix`

Returns Binned IntensityMatrix object

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

import_leco_csv (*file_name*)

Imports data in LECO CSV format.

Parameters **file_name** (`Union[str, Path, PathLike]`) – Path of the file to read.

Return type *IntensityMatrix*

Returns Data as an *IntensityMatrix*.

Authors Andrew Isaac, Dominic Davis-Foster (pathlib support)

6.3.9 `pyms.IonChromatogram`

Classes to model a GC-MS Ion Chromatogram.

Classes:

<code>BasePeakChromatogram(intensity_list, time_list)</code>	Models a base peak chromatogram (BPC).
<code>ExtractedIonChromatogram(intensity_list, ...)</code>	Models an extracted ion chromatogram (EIC).
<code>IonChromatogram(intensity_list, time_list[, ...])</code>	Models an ion chromatogram.

class `BasePeakChromatogram` (*intensity_list, time_list*)

Bases: *IonChromatogram*

Models a base peak chromatogram (BPC).

An ion chromatogram is a set of intensities as a function of retention time. This can be either m/z channel intensities (for example, ion chromatograms at $m/z = 65$), or cumulative intensities over all measured m/z . In the latter case the ion chromatogram is total ion chromatogram (TIC).

Parameters

- **intensity_list** (`Union[Sequence[float], ndarray]`) – Ion chromatogram intensity values
- **time_list** (`Sequence[float]`) – A list of ion chromatogram retention times

Authors Lewis Lee, Vladimir Likic, Dominic Davis-Foster (type assertions and properties)

New in version 2.3.0.

Methods:

<code>__copy__()</code>	Returns a new <i>IonChromatogram</i> containing a copy of the data in this object.
<code>__eq__(other)</code>	Return whether this <i>IonChromatogram</i> object is equal to another object.
<code>__len__()</code>	Returns the length of the <i>IonChromatogram</i> object.
<code>__sub__(other)</code>	Subtracts another IC from the current one (in place).
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>get_index_at_time(time)</code>	Returns the nearest index corresponding to the given time.
<code>get_intensity_at_index(ix)</code>	Returns the intensity at the given index.
<code>get_time_at_index(ix)</code>	Returns time at given index.

continues on next page

Table 41 – continued from previous page

<code>is_bpc()</code>	Returns whether the ion chromatogram is a base peak chromatogram (BPC).
<code>is_eic()</code>	Returns whether the ion chromatogram is an extracted ion chromatogram (EIC).
<code>is_tic()</code>	Returns whether the ion chromatogram is a total ion chromatogram (TIC) or extracted ion chromatogram (EIC).
<code>write(file_name[, minutes, formatting])</code>	Writes the ion chromatogram to the specified file.

Attributes:

<code>intensity_array</code>	Returns a copy of the intensity array.
<code>intensity_array_list</code>	Returns a copy of the intensity array as a list of lists of floats.
<code>intensity_matrix</code>	Returns a copy of the intensity matrix.
<code>mass</code>	Returns the m/z channel of the IC.
<code>matrix_list</code>	Returns the intensity matrix as a list of lists of floats.
<code>time_list</code>	Returns a copy of the time list.
<code>time_step</code>	Returns the time step.

`__copy__()`

Returns a new IonChromatogram containing a copy of the data in this object.

Return type `IonChromatogram`

`__eq__(other)`

Return whether this IonChromatogram object is equal to another object.

Parameters **other** (`Any`) – The other object to test equality with.

Return type `bool`

`__len__()`

Returns the length of the IonChromatogram object.

Authors Lewis Lee, Vladimir Likic

Return type `int`

`__sub__(other)`

Subtracts another IC from the current one (in place).

Parameters **other** (`IonChromatogram`) – Another IC

Return type `IonChromatogram`

`dump(file_name, protocol=3)`

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – Filename to save the dump as.
- **protocol** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

get_index_at_time (*time*)

Returns the nearest index corresponding to the given time.

Parameters *time* (*float*) – Time in seconds

Return type *int*

Returns Nearest index corresponding to given time

Authors Lewis Lee, Tim Erwin, Vladimir Likic

Changed in version 2.3.0: Now returns -1 if no index is found.

get_intensity_at_index (*ix*)

Returns the intensity at the given index.

Parameters *ix* (*int*) – An index.

Authors Lewis Lee, Vladimir Likic

Return type *float*

get_time_at_index (*ix*)

Returns time at given index.

Parameters *ix* (*int*)

Authors Lewis Lee, Vladimir Likic

Return type *float*

property intensity_array

Returns a copy of the intensity array.

Return type *ndarray*

Returns Matrix of intensity values.

Authors Andrew Isaac, Lewis Lee

property intensity_array_list

Returns a copy of the intensity array as a list of lists of floats.

Return type *List[List[float]]*

Returns Matrix of intensity values.

Author Andrew Isaac

property intensity_matrix

Returns a copy of the intensity matrix.

Return type *ndarray*

Returns Matrix of intensity values.

Author Andrew Isaac

static is_bpc()

Returns whether the ion chromatogram is a base peak chromatogram (BPC).

Return type `bool`

static is_eic()

Returns whether the ion chromatogram is an extracted ion chromatogram (EIC).

New in version 2.3.0.

Return type `bool`

is_tic()

Returns whether the ion chromatogram is a total ion chromatogram (TIC) or extracted ion chromatogram (EIC).

Authors Lewis Lee, Vladimir Likic

Return type `bool`

property mass

Returns the m/z channel of the IC.

Author Sean O’Callaghan

Return type `Optional[float]`

property matrix_list

Returns the intensity matrix as a list of lists of floats.

Return type `ndarray`

Returns Matrix of intensity values

Author Andrew Isaac

property time_list

Returns a copy of the time list.

Return type `List[float]`

Returns List of retention times

Authors Andrew Isaac, Lewis Lee, Vladimir Likic

property time_step

Returns the time step.

Authors Lewis Lee, Vladimir Likic

Return type `float`

write (*file_name*, *minutes=False*, *formatting=True*)

Writes the ion chromatogram to the specified file.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – The name of the output file

- **minutes** (`bool`) – A boolean value indicating whether to write time in minutes. Default `False`.
- **formatting** (`bool`) – Whether to format the numbers in the output. Default `True`.

Authors Lewis Lee, Vladimir Likic, Dominic Davis-Foster (pathlib support)

class `ExtractedIonChromatogram` (*intensity_list, time_list, masses*)

Bases: `IonChromatogram`

Models an extracted ion chromatogram (EIC).

An ion chromatogram is a set of intensities as a function of retention time. This can be either m/z channel intensities (for example, ion chromatograms at $m/z = 65$), or cumulative intensities over all measured m/z . In the latter case the ion chromatogram is total ion chromatogram (TIC).

Parameters

- **intensity_list** (`Union[Sequence[float], ndarray]`) – Ion chromatogram intensity values
- **time_list** (`Sequence[float]`) – A list of ion chromatogram retention times
- **masses** (`Sequence[float]`) – List of extracted masses in the EIC.

Authors Lewis Lee, Vladimir Likic, Dominic Davis-Foster (type assertions and properties)

New in version 2.3.0.

Methods:

<code>__copy__()</code>	Returns a new <code>IonChromatogram</code> containing a copy of the data in this object.
<code>__eq__(other)</code>	Return whether this <code>IonChromatogram</code> object is equal to another object.
<code>__len__()</code>	Returns the length of the <code>IonChromatogram</code> object.
<code>__sub__(other)</code>	Subtracts another IC from the current one (in place).
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>get_index_at_time(time)</code>	Returns the nearest index corresponding to the given time.
<code>get_intensity_at_index(ix)</code>	Returns the intensity at the given index.
<code>get_time_at_index(ix)</code>	Returns time at given index.
<code>is_bpc()</code>	Returns whether the ion chromatogram is a base peak chromatogram (BPC).
<code>is_eic()</code>	Returns whether the ion chromatogram is an extracted ion chromatogram (EIC).
<code>is_tic()</code>	Returns whether the ion chromatogram is a total ion chromatogram (TIC) or extracted ion chromatogram (EIC).
<code>write(file_name[, minutes, formatting])</code>	Writes the ion chromatogram to the specified file.

Attributes:

<code>intensity_array</code>	Returns a copy of the intensity array.
<code>intensity_array_list</code>	Returns a copy of the intensity array as a list of lists of floats.

continues on next page

Table 44 – continued from previous page

<i>intensity_matrix</i>	Returns a copy of the intensity matrix.
<i>mass</i>	Returns the m/z channel of the IC.
<i>masses</i>	List of extracted masses in the EIC.
<i>matrix_list</i>	Returns the intensity matrix as a list of lists of floats.
<i>time_list</i>	Returns a copy of the time list.
<i>time_step</i>	Returns the time step.

__copy__()

Returns a new IonChromatogram containing a copy of the data in this object.

Return type *IonChromatogram*

__eq__() (*other*)

Return whether this IonChromatogram object is equal to another object.

Parameters **other** (*Any*) – The other object to test equality with.

Return type *bool*

__len__()

Returns the length of the IonChromatogram object.

Authors Lewis Lee, Vladimir Likic

Return type *int*

__sub__() (*other*)

Subtracts another IC from the current one (in place).

Parameters **other** (*IonChromatogram*) – Another IC

Return type *IonChromatogram*

dump() (*file_name*, *protocol=3*)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (*Union[str, Path, PathLike]*) – Filename to save the dump as.
- **protocol** (*int*) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

get_index_at_time() (*time*)

Returns the nearest index corresponding to the given time.

Parameters **time** (*float*) – Time in seconds

Return type *int*

Returns Nearest index corresponding to given time

Authors Lewis Lee, Tim Erwin, Vladimir Likic

Changed in version 2.3.0: Now returns -1 if no index is found.

get_intensity_at_index (*ix*)

Returns the intensity at the given index.

Parameters *ix* (*int*) – An index.

Authors Lewis Lee, Vladimir Likic

Return type *float*

get_time_at_index (*ix*)

Returns time at given index.

Parameters *ix* (*int*)

Authors Lewis Lee, Vladimir Likic

Return type *float*

property intensity_array

Returns a copy of the intensity array.

Return type *ndarray*

Returns Matrix of intensity values.

Authors Andrew Isaac, Lewis Lee

property intensity_array_list

Returns a copy of the intensity array as a list of lists of floats.

Return type *List[List[float]]*

Returns Matrix of intensity values.

Author Andrew Isaac

property intensity_matrix

Returns a copy of the intensity matrix.

Return type *ndarray*

Returns Matrix of intensity values.

Author Andrew Isaac

static is_bpc ()

Returns whether the ion chromatogram is a base peak chromatogram (BPC).

New in version 2.3.0.

Return type *bool*

static is_eic ()

Returns whether the ion chromatogram is an extracted ion chromatogram (EIC).

Return type *bool*

is_tic ()

Returns whether the ion chromatogram is a total ion chromatogram (TIC) or extracted ion chromatogram (EIC).

Authors Lewis Lee, Vladimir Likic

Return type `bool`

property mass

Returns the m/z channel of the IC.

Author Sean O’Callaghan

Return type `Optional[float]`

property masses

List of extracted masses in the EIC.

Return type `Tuple[float, ...]`

property matrix_list

Returns the intensity matrix as a list of lists of floats.

Return type `ndarray`

Returns Matrix of intensity values

Author Andrew Isaac

property time_list

Returns a copy of the time list.

Return type `List[float]`

Returns List of retention times

Authors Andrew Isaac, Lewis Lee, Vladimir Likic

property time_step

Returns the time step.

Authors Lewis Lee, Vladimir Likic

Return type `float`

write (*file_name*, *minutes=False*, *formatting=True*)

Writes the ion chromatogram to the specified file.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – The name of the output file
- **minutes** (`bool`) – A boolean value indicating whether to write time in minutes. Default `False`.
- **formatting** (`bool`) – Whether to format the numbers in the output. Default `True`.

Authors Lewis Lee, Vladimir Likic, Dominic Davis-Foster (pathlib support)

class IonChromatogram (*intensity_list*, *time_list*, *mass=None*)

Bases: `pymsBaseClass`, `TimeListMixin`, `IntensityArrayMixin`, `GetIndexTimeMixin`

Models an ion chromatogram.

An ion chromatogram is a set of intensities as a function of retention time. This can be either m/z channel intensities (for example, ion chromatograms at $m/z = 65$), or cumulative intensities over all measured m/z . In the latter case the ion chromatogram is total ion chromatogram (TIC).

The nature of an IonChromatogram object can be revealed by inspecting the value of the attribute 'mass'. This is set to the m/z value of the ion chromatogram, or to `None` for TIC.

Parameters

- **intensity_list** (`Union[Sequence[float], ndarray]`) – Ion chromatogram intensity values
- **time_list** (`Sequence[float]`) – A list of ion chromatogram retention times
- **mass** (`Optional[float]`) – Mass of ion chromatogram (`None` if TIC). Default `None`.

Authors Lewis Lee, Vladimir Likic, Dominic Davis-Foster (type assertions and properties)

Changed in version 2.3.0: The `ia` parameter was renamed to `intensity_list`.

Methods:

<code>__copy__()</code>	Returns a new IonChromatogram containing a copy of the data in this object.
<code>__eq__(other)</code>	Return whether this IonChromatogram object is equal to another object.
<code>__len__()</code>	Returns the length of the IonChromatogram object.
<code>__sub__(other)</code>	Subtracts another IC from the current one (in place).
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>get_index_at_time(time)</code>	Returns the nearest index corresponding to the given time.
<code>get_intensity_at_index(ix)</code>	Returns the intensity at the given index.
<code>get_time_at_index(ix)</code>	Returns time at given index.
<code>is_bpc()</code>	Returns whether the ion chromatogram is a base peak chromatogram (BPC).
<code>is_eic()</code>	Returns whether the ion chromatogram is an extracted ion chromatogram (EIC).
<code>is_tic()</code>	Returns whether the ion chromatogram is a total ion chromatogram (TIC) or extracted ion chromatogram (EIC).
<code>write(file_name[, minutes, formatting])</code>	Writes the ion chromatogram to the specified file.

Attributes:

<code>intensity_array</code>	Returns a copy of the intensity array.
<code>intensity_array_list</code>	Returns a copy of the intensity array as a list of lists of floats.
<code>intensity_matrix</code>	Returns a copy of the intensity matrix.
<code>mass</code>	Returns the m/z channel of the IC.
<code>matrix_list</code>	Returns the intensity matrix as a list of lists of floats.
<code>time_list</code>	Returns a copy of the time list.
<code>time_step</code>	Returns the time step.

`__copy__()`

Returns a new IonChromatogram containing a copy of the data in this object.

Return type `IonChromatogram`

`__eq__` (*other*)

Return whether this IonChromatogram object is equal to another object.

Parameters **other** (*Any*) – The other object to test equality with.

Return type `bool`

`__len__` ()

Returns the length of the IonChromatogram object.

Authors Lewis Lee, Vladimir Likic

Return type `int`

`__sub__` (*other*)

Subtracts another IC from the current one (in place).

Parameters **other** (*IonChromatogram*) – Another IC

Return type `IonChromatogram`

dump (*file_name*, *protocol=3*)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – Filename to save the dump as.
- **protocol** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

get_index_at_time (*time*)

Returns the nearest index corresponding to the given time.

Parameters **time** (`float`) – Time in seconds

Return type `int`

Returns Nearest index corresponding to given time

Authors Lewis Lee, Tim Erwin, Vladimir Likic

Changed in version 2.3.0: Now returns -1 if no index is found.

get_intensity_at_index (*ix*)

Returns the intensity at the given index.

Parameters **ix** (`int`) – An index.

Authors Lewis Lee, Vladimir Likic

Return type `float`

get_time_at_index (*ix*)

Returns time at given index.

Parameters `ix` (`int`)

Authors Lewis Lee, Vladimir Likic

Return type `float`

property `intensity_array`

Returns a copy of the intensity array.

Return type `ndarray`

Returns Matrix of intensity values.

Authors Andrew Isaac, Lewis Lee

property `intensity_array_list`

Returns a copy of the intensity array as a list of lists of floats.

Return type `List[List[float]]`

Returns Matrix of intensity values.

Author Andrew Isaac

property `intensity_matrix`

Returns a copy of the intensity matrix.

Return type `ndarray`

Returns Matrix of intensity values.

Author Andrew Isaac

static `is_bpc()`

Returns whether the ion chromatogram is a base peak chromatogram (BPC).

New in version 2.3.0.

Return type `bool`

static `is_eic()`

Returns whether the ion chromatogram is an extracted ion chromatogram (EIC).

New in version 2.3.0.

Return type `bool`

is_tic()

Returns whether the ion chromatogram is a total ion chromatogram (TIC) or extracted ion chromatogram (EIC).

Authors Lewis Lee, Vladimir Likic

Return type `bool`

property `mass`

Returns the m/z channel of the IC.

Author Sean O'Callaghan

Return type `Optional[float]`

property matrix_list

Returns the intensity matrix as a list of lists of floats.

Return type `ndarray`

Returns Matrix of intensity values

Author Andrew Isaac

property time_list

Returns a copy of the time list.

Return type `List[float]`

Returns List of retention times

Authors Andrew Isaac, Lewis Lee, Vladimir Likic

property time_step

Returns the time step.

Authors Lewis Lee, Vladimir Likic

Return type `float`

write (*file_name*, *minutes=False*, *formatting=True*)

Writes the ion chromatogram to the specified file.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – The name of the output file
- **minutes** (`bool`) – A boolean value indicating whether to write time in minutes. Default `False`.
- **formatting** (`bool`) – Whether to format the numbers in the output. Default `True`.

Authors Lewis Lee, Vladimir Likic, Dominic Davis-Foster (pathlib support)

6.3.10 `pym.s.json`

Custom JSON Encoder to support PyMassSpec classes.

Classes:

<code>PyMassSpecEncoder(*args, **kwargs)</code>	Custom JSON Encoder to support PyMassSpec classes.
---	--

class PyMassSpecEncoder (**args, **kwargs*)

Bases: `JSONEncoder`

Custom JSON Encoder to support PyMassSpec classes.

Methods:

<code>default(o)</code>	Implement this method in a subclass such that it returns a serializable object for <code>o</code> , or calls the base implementation (to raise a <code>TypeError</code>).
<code>encode(o)</code>	Return a JSON string representation of a Python data structure.
<code>iterencode(o[, _one_shot])</code>	Encode the given object and yield each string representation as available.

default (o)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement `default` like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

encode (o)

Return a JSON string representation of a Python data structure.

```
>>> from json.encoder import JSONEncoder
>>> JSONEncoder().encode({"foo": ["bar", "baz"]})
'{"foo": ["bar", "baz"]}'
```

Return type `Any`

iterencode (o, _one_shot=False)

Encode the given object and yield each string representation as available.

For example:

```
for chunk in JSONEncoder().iterencode(bigobject):
    mysocket.write(chunk)
```

Return type `Iterator[str]`

6.3.11 `pyms.Mixins`

Mixins for PyMassSpec Classes.

Classes:

<code>GetIndexTimeMixin()</code>	
<code>IntensityArrayMixin()</code>	
<code>MassListMixin()</code>	Mixin class to add the <code>mass_list</code> property, which returns a copy of the internal <code>_mass_list</code> attribute.
<code>MaxMinMassMixin()</code>	Mixin class to add the <code>min_mass</code> and <code>max_mass</code> properties, which provide read-only access to the internal <code>_min_mass</code> and <code>_max_mass</code> attributes.
<code>TimeListMixin()</code>	Mixin class to add the <code>time_list</code> property, which returns a copy of the internal <code>_time_list</code> attribute.

class `GetIndexTimeMixin`

Bases: `object`

Methods:

<code>get_index_at_time(time)</code>	Returns the nearest index corresponding to the given time.
<code>get_time_at_index(ix)</code>	Returns time at given index.

get_index_at_time (*time*)

Returns the nearest index corresponding to the given time.

Parameters `time` (`float`) – Time in seconds

Return type `int`

Returns Nearest index corresponding to given time

Authors Lewis Lee, Tim Erwin, Vladimir Likic

Changed in version 2.3.0: Now returns `-1` if no index is found.

get_time_at_index (*ix*)

Returns time at given index.

Parameters `ix` (`int`)

Authors Lewis Lee, Vladimir Likic

Return type `float`

class `IntensityArrayMixin`

Bases: `object`

Attributes:

<code>intensity_array</code>	Returns a copy of the intensity array.
------------------------------	--

continues on next page

Table 51 – continued from previous page

<i>intensity_array_list</i>	Returns a copy of the intensity array as a list of lists of floats.
<i>intensity_matrix</i>	Returns a copy of the intensity matrix.
<i>matrix_list</i>	Returns the intensity matrix as a list of lists of floats.

property intensity_array

Returns a copy of the intensity array.

Return type `ndarray`**Returns** Matrix of intensity values.**Authors** Andrew Isaac, Lewis Lee**property intensity_array_list**

Returns a copy of the intensity array as a list of lists of floats.

Return type `List[List[float]]`**Returns** Matrix of intensity values.**Author** Andrew Isaac**property intensity_matrix**

Returns a copy of the intensity matrix.

Return type `ndarray`**Returns** Matrix of intensity values.**Author** Andrew Isaac**property matrix_list**

Returns the intensity matrix as a list of lists of floats.

Return type `ndarray`**Returns** Matrix of intensity values**Author** Andrew Isaac**class MassListMixin**Bases: *MaxMinMassMixin*Mixin class to add the `mass_list` property, which returns a copy of the internal `_mass_list` attribute.**Attributes:**

<i>mass_list</i>	Returns a list of the masses.
------------------	-------------------------------

property mass_list

Returns a list of the masses.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic**Return type** `List[float]`

class MaxMinMassMixinBases: `object`

Mixin class to add the `min_mass` and `max_mass` properties, which provide read-only access to the internal `_min_mass` and `_max_mass` attributes.

Attributes:

<code>max_mass</code>	Returns the maximum m/z value in the spectrum.
<code>min_mass</code>	Returns the minimum m/z value in the spectrum.

property max_massReturns the maximum m/z value in the spectrum.**Author** Andrew Isaac**Return type** `Optional[float]`**property min_mass**Returns the minimum m/z value in the spectrum.**Author** Andrew Isaac**Return type** `Optional[float]`**class TimeListMixin**Bases: `object`

Mixin class to add the `time_list` property, which returns a copy of the internal `_time_list` attribute.

Attributes:

<code>time_list</code>	Returns a copy of the time list.
------------------------	----------------------------------

property time_list

Returns a copy of the time list.

Return type `List[float]`**Returns** List of retention times**Authors** Andrew Isaac, Lewis Lee, Vladimir Likic**6.3.12 pyms.Spectrum**

Classes to model Mass Spectra and Scans.

Classes:

<code>CompositeMassSpectrum(mass_list, intensity_list)</code>	intensity_list	Represents a composite mass spectrum.
<code>MassSpectrum(mass_list, intensity_list)</code>		Models a binned mass spectrum.
<code>Scan(mass_list, intensity_list)</code>		Generic object for a single Scan's raw data.

Data:

<code>_C</code>	Invariant <code>TypeVar</code> bound to <code>pym.spectrum.CompositeMassSpectrum</code> .
<code>_M</code>	Invariant <code>TypeVar</code> bound to <code>pym.spectrum.MassSpectrum</code> .
<code>_S</code>	Invariant <code>TypeVar</code> bound to <code>pym.spectrum.Scan</code> .

Functions:

<code>array_as_numeric(array)</code>	Convert the given numpy array to a numeric data type.
<code>normalize_mass_spec(mass_spec[, ...])</code>	Normalize the intensities in the given Mass Spectrum to values between 0 and <code>max_intensity</code> , which by default is 100.0.

class `CompositeMassSpectrum` (*mass_list*, *intensity_list*)

Bases: `MassSpectrum`

Represents a composite mass spectrum.

Parameters

- **mass_list** (`Union[Sequence[float], ndarray]`) – mass values
- **intensity_list** (`Union[Sequence[float], ndarray]`) – intensity values

Author Dominic Davis-Foster

Methods:

<code>__copy__()</code>	Returns a copy of the object.
<code>__eq__(other)</code>	Return whether this object is equal to another object.
<code>__len__()</code>	Returns the length of the object.
<code>crop([min_mz, max_mz, inplace])</code>	Crop the Mass Spectrum between the given mz values.
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>from_dict(dictionary)</code>	Create a <code>Scan</code> from a dictionary.
<code>from_jcamp(file_name)</code>	Create a <code>MassSpectrum</code> from a JCAMP-DX file.
<code>from_mz_int_pairs(mz_int_pairs)</code>	Construct a <code>MassSpectrum</code> from a list of (m/z, intensity) tuples.
<code>from_spectra(spectra)</code>	Construct a <code>CompositeMassSpectrum</code> from multiple <code>MassSpectrum</code> objects.
<code>get_intensity_for_mass(mass)</code>	Returns the intensity for the given mass.
<code>get_mass_for_intensity(intensity)</code>	Returns the mass for the given intensity.
<code>icrop([min_index, max_index, inplace])</code>	Crop the Mass Spectrum between the given indices.
<code>iter_peaks()</code>	Iterate over the peaks in the mass spectrum.
<code>n_largest_peaks(n)</code>	Returns the indices of the <code>n</code> largest peaks in the Mass Spectrum.

Attributes:

<code>intensity_list</code>	Returns a copy of the intensity list.
<code>mass_list</code>	Returns a list of the masses.
<code>mass_spec</code>	Returns the intensity list.
<code>max_mass</code>	Returns the maximum m/z value in the spectrum.
<code>min_mass</code>	Returns the minimum m/z value in the spectrum.
<code>size</code>	The number of mass spectra combined to create this composite spectrum.

`__copy__()`

Returns a copy of the object.

Return type `Scan`

`__eq__(other)`

Return whether this object is equal to another object.

Parameters `other` (Any) – The other object to test equality with.

Return type `bool`

`__len__()`

Returns the length of the object.

Authors Andrew Isaac, Qiao Wang, Vladimir Likic

Return type `int`

`crop(min_mz=None, max_mz=None, inplace=False)`

Crop the Mass Spectrum between the given mz values.

Parameters

- **min_mz** (Optional[float]) – The minimum mz for the new mass spectrum. Default `None`.
- **max_mz** (Optional[float]) – The maximum mz for the new mass spectrum. Default `None`.
- **inplace** (bool) – Whether the cropping should be applied this instance or to a copy (default behaviour). Default `False`.

Return type `~_M`

Returns The cropped Mass Spectrum

`dump(file_name, protocol=3)`

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (Union[str, Path, PathLike]) – Filename to save the dump as.
- **protocol** (int) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

`classmethod from_dict(dictionary)`

Create a `Scan` from a dictionary.

The dictionary's keys must match the arguments taken by the class's constructor.

Parameters `dictionary` (`Mapping`)

Return type `~_S`

classmethod `from_jcamp` (`file_name`)

Create a `MassSpectrum` from a JCAMP-DX file.

Parameters `file_name` (`Union[str, Path, PathLike]`) – Path of the file to read.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic, David Kainer, Dominic Davis-Foster

Return type `~_M`

classmethod `from_mz_int_pairs` (`mz_int_pairs`)

Construct a `MassSpectrum` from a list of (m/z, intensity) tuples.

Parameters `mz_int_pairs` (`Sequence[Tuple[float, float]]`)

Return type `~_M`

classmethod `from_spectra` (`spectra`)

Construct a `CompositeMassSpectrum` from multiple `MassSpectrum` objects.

If no `MassSpectrum` objects are given an empty `CompositeMassSpectrum` is returned.

Parameters `spectra` (`Iterable[MassSpectrum]`)

Return type `~_C`

get_intensity_for_mass (`mass`)

Returns the intensity for the given mass.

Parameters `mass` (`float`)

Return type `float`

get_mass_for_intensity (`intensity`)

Returns the mass for the given intensity. If more than one mass has the given intensity, the first mass is returned.

Parameters `intensity` (`float`)

Return type `float`

icrop (`min_index=0, max_index=-1, inplace=False`)

Crop the Mass Spectrum between the given indices.

Parameters

- **min_index** (`int`) – The minimum index for the new mass spectrum. Default 0.
- **max_index** (`int`) – The maximum index for the new mass spectrum. Default -1.
- **inplace** (`bool`) – Whether the cropping should be applied this instance or to a copy (default behaviour). Default `False`.

Return type `~_M`

Returns The cropped Mass Spectrum

property intensity_list

Returns a copy of the intensity list.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List`

iter_peaks()

Iterate over the peaks in the mass spectrum.

Return type `Iterator[Tuple[float, float]]`

property mass_list

Returns a list of the masses.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List[float]`

property mass_spec

Returns the intensity list.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List`

property max_mass

Returns the maximum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

property min_mass

Returns the minimum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

n_largest_peaks(n)

Returns the indices of the n largest peaks in the Mass Spectrum.

Parameters n (`int`) – The number of peaks to return the indices for.

Return type `List[int]`

size = 1

Type: `int`

The number of mass spectra combined to create this composite spectrum.

class MassSpectrum(mass_list, intensity_list)

Bases: `Scan`

Models a binned mass spectrum.

Parameters

- **mass_list** (`Union[Sequence[float], ndarray]`) – mass values
- **intensity_list** (`Union[Sequence[float], ndarray]`) – intensity values

Authors Andrew Isaac, Qiao Wang, Vladimir Likic, Dominic Davis-Foster

Methods:

<code>__copy__()</code>	Returns a copy of the object.
<code>__eq__(other)</code>	Return whether this object is equal to another object.
<code>__len__()</code>	Returns the length of the object.
<code>crop([min_mz, max_mz, inplace])</code>	Crop the Mass Spectrum between the given mz values.
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>from_dict(dictionary)</code>	Create a <code>Scan</code> from a dictionary.
<code>from_jcamp(file_name)</code>	Create a <code>MassSpectrum</code> from a JCAMP-DX file.
<code>from_mz_int_pairs(mz_int_pairs)</code>	Construct a <code>MassSpectrum</code> from a list of (m/z, intensity) tuples.
<code>get_intensity_for_mass(mass)</code>	Returns the intensity for the given mass.
<code>get_mass_for_intensity(intensity)</code>	Returns the mass for the given intensity.
<code>icrop([min_index, max_index, inplace])</code>	Crop the Mass Spectrum between the given indices.
<code>iter_peaks()</code>	Iterate over the peaks in the mass spectrum.
<code>n_largest_peaks(n)</code>	Returns the indices of the n largest peaks in the Mass Spectrum.

Attributes:

<code>intensity_list</code>	Returns a copy of the intensity list.
<code>mass_list</code>	Returns a list of the masses.
<code>mass_spec</code>	Returns the intensity list.
<code>max_mass</code>	Returns the maximum <i>m/z</i> value in the spectrum.
<code>min_mass</code>	Returns the minimum <i>m/z</i> value in the spectrum.

`__copy__()`
Returns a copy of the object.

Return type `Scan`

`__eq__(other)`
Return whether this object is equal to another object.

Parameters `other` (`Any`) – The other object to test equality with.

Return type `bool`

`__len__()`
Returns the length of the object.

Authors Andrew Isaac, Qiao Wang, Vladimir Likic

Return type `int`

`crop` (`min_mz=None`, `max_mz=None`, `inplace=False`)

Crop the Mass Spectrum between the given mz values.

Parameters

- **min_mz** (Optional[float]) – The minimum mz for the new mass spectrum. Default `None`.
- **max_mz** (Optional[float]) – The maximum mz for the new mass spectrum. Default `None`.
- **inplace** (bool) – Whether the cropping should be applied this instance or to a copy (default behaviour). Default `False`.

Return type `~_M`

Returns The cropped Mass Spectrum

dump (*file_name*, *protocol=3*)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (Union[str, Path, PathLike]) – Filename to save the dump as.
- **protocol** (int) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

classmethod from_dict (*dictionary*)

Create a `Scan` from a dictionary.

The dictionary's keys must match the arguments taken by the class's constructor.

Parameters **dictionary** (Mapping)

Return type `~_S`

classmethod from_jcamp (*file_name*)

Create a MassSpectrum from a JCAMP-DX file.

Parameters **file_name** (Union[str, Path, PathLike]) – Path of the file to read.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic, David Kainer, Dominic Davis-Foster

Return type `~_M`

classmethod from_mz_int_pairs (*mz_int_pairs*)

Construct a MassSpectrum from a list of (m/z, intensity) tuples.

Parameters **mz_int_pairs** (Sequence[Tuple[float, float]])

Return type `~_M`

get_intensity_for_mass (*mass*)

Returns the intensity for the given mass.

Parameters **mass** (float)

Return type float

get_mass_for_intensity (*intensity*)

Returns the mass for the given intensity. If more than one mass has the given intensity, the first mass is returned.

Parameters `intensity` (`float`)

Return type `float`

`icrop` (`min_index=0, max_index=-1, inplace=False`)
Crop the Mass Spectrum between the given indices.

Parameters

- **`min_index`** (`int`) – The minimum index for the new mass spectrum. Default 0.
- **`max_index`** (`int`) – The maximum index for the new mass spectrum. Default -1.
- **`inplace`** (`bool`) – Whether the cropping should be applied this instance or to a copy (default behaviour). Default `False`.

Return type `~_M`

Returns The cropped Mass Spectrum

property `intensity_list`
Returns a copy of the intensity list.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List`

`iter_peaks` ()
Iterate over the peaks in the mass spectrum.

Return type `Iterator[Tuple[float, float]]`

property `mass_list`
Returns a list of the masses.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List[float]`

property `mass_spec`
Returns the intensity list.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List`

property `max_mass`
Returns the maximum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

property `min_mass`
Returns the minimum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

n_largest_peaks (*n*)

Returns the indices of the *n* largest peaks in the Mass Spectrum.

Parameters *n* (*int*) – The number of peaks to return the indices for.

Return type `List[int]`

class Scan (*mass_list*, *intensity_list*)

Bases: `pymBaseClass`, `MassListMixin`

Generic object for a single Scan's raw data.

Parameters

- **mass_list** (`Union[Sequence[float], ndarray]`) – A sequence of mass values
- **intensity_list** (`Union[Sequence[float], ndarray]`) – A sequence intensity values

Authors Andrew Isaac, Qiao Wang, Vladimir Likic, Dominic Davis-Foster

Methods:

<code>__copy__()</code>	Returns a copy of the object.
<code>__eq__(other)</code>	Return whether this object is equal to another object.
<code>__len__()</code>	Returns the length of the object.
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>from_dict(dictionary)</code>	Create a <code>Scan</code> from a dictionary.
<code>iter_peaks()</code>	Iterate over the peaks in the mass spectrum.

Attributes:

<code>intensity_list</code>	Returns a copy of the intensity list.
<code>mass_list</code>	Returns a list of the masses.
<code>mass_spec</code>	Returns the intensity list.
<code>max_mass</code>	Returns the maximum m/z value in the spectrum.
<code>min_mass</code>	Returns the minimum m/z value in the spectrum.

`__copy__()`

Returns a copy of the object.

Return type `Scan`

`__eq__(other)`

Return whether this object is equal to another object.

Parameters *other* (*Any*) – The other object to test equality with.

Return type `bool`

`__len__()`

Returns the length of the object.

Authors Andrew Isaac, Qiao Wang, Vladimir Likic

Return type `int`

dump (*file_name*, *protocol*=3)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – Filename to save the dump as.
- **protocol** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

classmethod from_dict (*dictionary*)

Create a `Scan` from a dictionary.

The dictionary's keys must match the arguments taken by the class's constructor.

Parameters **dictionary** (`Mapping`)

Return type `~_S`

property intensity_list

Returns a copy of the intensity list.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List`

iter_peaks ()

Iterate over the peaks in the mass spectrum.

Return type `Iterator[Tuple[float, float]]`

property mass_list

Returns a list of the masses.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List[float]`

property mass_spec

Returns the intensity list.

Authors Qiao Wang, Andrew Isaac, Vladimir Likic

Return type `List`

property max_mass

Returns the maximum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

property min_mass

Returns the minimum m/z value in the spectrum.

Author Andrew Isaac

Return type `Optional[float]`

_C = TypeVar(_C, bound=CompositeMassSpectrum)

Type: `TypeVar`

Invariant `TypeVar` bound to `pyms.Spectrum.CompositeMassSpectrum`.

_M = TypeVar(_M, bound=MassSpectrum)

Type: `TypeVar`

Invariant `TypeVar` bound to `pyms.Spectrum.MassSpectrum`.

_S = TypeVar(_S, bound=Scan)

Type: `TypeVar`

Invariant `TypeVar` bound to `pyms.Spectrum.Scan`.

array_as_numeric(array)

Convert the given numpy array to a numeric data type.

If the data in the array is already in a numeric data type no changes will be made.

If array is a python `Sequence` then it will first be converted to a numpy array.

Parameters `array` (`Union[Sequence, ndarray]`)

Return type `ndarray`

normalize_mass_spec(mass_spec, relative_to=None, inplace=False, max_intensity=100)

Normalize the intensities in the given Mass Spectrum to values between 0 and `max_intensity`, which by default is 100.0.

Parameters

- **mass_spec** (`MassSpectrum`) – The Mass Spectrum to normalize
- **relative_to** (`Optional[float]`) – The largest intensity in the original data set. If not `None` the intensities are computed relative to this value. If `None` the value is calculated from the mass spectrum. This can be useful when normalizing several mass spectra to each other. Default `None`.
- **inplace** (`bool`) – Whether the normalization should be applied to the `MassSpectrum` object given, or to a copy (default behaviour). Default `False`.
- **max_intensity** (`float`) – The maximum intensity in the normalized spectrum. If omitted the range 0-100.0 is used. If an integer the normalized intensities will be integers. Default 100.

Return type `MassSpectrum`

Returns The normalized mass spectrum

6.3.13 pyms.Noise

Table of Contents

- `pyms.Noise`
 - `pyms.Noise.Analysis`
 - `pyms.Noise.SavitzkyGolay`
 - `pyms.Noise.Window`

Noise processing functions.

`pyms.Noise.Analysis`

Noise analysis functions.

Functions:

<code>window_analyzer(ic[, window, n_windows, ...])</code>	A simple estimator of the signal noise based on randomly placed windows and median absolute deviation.
--	--

window_analyzer (*ic*, *window*=256, *n_windows*=1024, *rand_seed*=None)

A simple estimator of the signal noise based on randomly placed windows and median absolute deviation.

The noise value is estimated by repeatedly and picking random windows (of a specified width) and calculating median absolute deviation (MAD). The noise estimate is given by the minimum MAD.

Parameters

- **ic** (*IonChromatogram*)
- **window** (`Union[int, str]`) – Window width selection. Default 256.
- **n_windows** (`int`) – The number of windows to calculate. Default 1024.
- **rand_seed** (`Union[int, float, str, None]`) – Seed for random number generator. Default None.

Return type `float`

Returns The noise estimate.

Author Vladimir Likic

`pyms.Noise.SavitzkyGolay`

Savitzky-Golay noise filter.

Functions:

<code>savitzky_golay(ic[, window, degree])</code>	Applies Savitzky-Golay filter on an ion chromatogram.
<code>savitzky_golay_im(im[, window, degree])</code>	Applies Savitzky-Golay filter on Intensity Matrix.

savitzky_golay (*ic*, *window*=7, *degree*=2)

Applies Savitzky-Golay filter on an ion chromatogram.

Parameters

- **ic** (*IonChromatogram*) – The input ion chromatogram.
- **window** (`Union[int, str]`) – The window selection parameter. This can be an integer or time string. If an integer, taken as the number of points. If a string, must be the form '<NUMBER>s' or '<NUMBER>m', specifying a time in seconds or minutes, respectively. Default 7.
- **degree** (`int`) – degree of the fitting polynomial for the Savitzky-Golay filter. Default 2.

Return type *IonChromatogram*

Returns Smoothed ion chromatogram.

Authors Uwe Schmitt, Vladimir Likic, Dominic Davis-Foster

savitzky_golay_im(*im*, *window*=7, *degree*=2)

Applies Savitzky-Golay filter on Intensity Matrix.

Simply wraps around the Savitzky Golay function above.

Parameters

- **im** (*BaseIntensityMatrix*)
- **window** (*Union[int, str]*) – The window selection parameter. Default 7.
- **degree** (*int*) – degree of the fitting polynomial for the Savitzky-Golay filter. Default 2.

Returns Smoothed IntensityMatrix.

Return type *BaseIntensityMatrix*

Authors Sean O’Callaghan, Vladimir Likic, Dominic Davis-Foster

pyms.Noise.Window

Moving window noise filter.

Functions:

<code>window_smooth(ic[, window, use_median])</code>	Applies window smoothing on ion chromatogram.
<code>window_smooth_im(im[, window, use_median])</code>	Applies window smoothing on Intensity Matrix.

window_smooth(*ic*, *window*=3, *use_median*=False)

Applies window smoothing on ion chromatogram.

Parameters

- **ic** (*IonChromatogram*)
- **window** (*Union[int, str]*) – The window selection parameter. This can be an integer or time string. If an integer, taken as the number of points. If a string, must be in the form '<NUMBER>s' or '<NUMBER>m', specifying a time in seconds or minutes, respectively Default 3.
- **use_median** (*bool*) – Whether to use the the mean or median window smoothing. Default False.

Return type *IonChromatogram*

Returns Smoothed ion chromatogram

Authors Vladimir Likic, Dominic Davis-Foster (type assertions)

window_smooth_im(*im*, *window*=3, *use_median*=False)

Applies window smoothing on Intensity Matrix.

Simply wraps around the window smooth function above.

Parameters

- **im** (*BaseIntensityMatrix*)

- **window** (`Union[int, str]`) – The window selection parameter. Default 3.
- **use_median** (`bool`) – If `True` median window smoothing will be used. If `False` mean window smoothing will be used. Default `False`.

Returns Smoothed Intensity Matrix

Return type `BaseIntensityMatrix`

Authors Sean O’Callaghan, Vladimir Likic

6.3.14 `pyms.Peak`

Table of Contents

- `pyms.Peak`
 - `pyms.Peak.Class`
 - `pyms.Peak.Function`
 - `pyms.Peak.List`
 - * `pyms.Peak.List.Function`
 - * `pyms.Peak.List.IO`

Functions for modelling signal peaks.

`pyms.Peak.Class`

Provides a class to model signal peak.

Classes:

<code>AbstractPeak([rt, minutes, outlier])</code>	Models a signal peak.
<code>ICPeak([rt, mass, minutes, outlier])</code>	Subclass of <code>Peak</code> representing a peak in an ion chromatogram for a single mass.
<code>Peak()</code>	Subclass of <code>Peak</code> representing a peak in a mass spectrum.

class `AbstractPeak` (`rt=0.0, minutes=False, outlier=False`)

Bases: `pymsBaseClass`

Models a signal peak.

Parameters

- **rt** (`Union[int, float]`) – Retention time. Default 0.0.
- **minutes** (`bool`) – Retention time units flag. If `True`, retention time is in minutes; if `False` retention time is in seconds. Default `False`.
- **outlier** (`bool`) – Whether the peak is an outlier. Default `False`.

Authors Vladimir Likic, Andrew Isaac, Dominic Davis-Foster (type assertions and properties), David Kainer (outlier flag)

New in version 2.3.0.

Attributes:

<i>UID</i>	Return the unique peak ID (UID), either:
<i>area</i>	The area under the peak.
<i>bounds</i>	The peak boundaries in points.
<i>ion_areas</i>	Returns a copy of the ion areas dict.
<i>rt</i>	The retention time of the peak, in seconds.

Methods:

<code>__eq__(other)</code>	Return whether this Peak object is equal to another object.
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>get_ion_area(ion)</code>	Returns the area of a single ion chromatogram under the peak.
<code>make_UID()</code>	Create a unique peak ID (UID).
<code>set_bounds(left, apex, right)</code>	Sets peak boundaries in points.
<code>set_ion_area(ion, area)</code>	Sets the area for a single ion.

property UID

Return the unique peak ID (UID), either:

- Integer masses of top two intensities and their ratio (as `Mass1-Mass2-Ratio*100`); or
- the single mass as an integer and the retention time.

Return type `str`

Returns UID string

Author Andrew Isaac

`__eq__(other)`

Return whether this Peak object is equal to another object.

Parameters `other` – The other object to test equality with.

Return type `bool`

property area

The area under the peak.

Author Andrew Isaac

Return type `Optional[float]`

property bounds

The peak boundaries in points.

Return type `Optional[Tuple[int, int, int]]`

Returns A 3-element tuple containing the left, apex, and right peak boundaries in points. Left and right are offsets.

Author Andrew Isaac

dump (*file_name*, *protocol=3*)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – Filename to save the dump as.
- **protocol** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

get_ion_area (*ion*)

Returns the area of a single ion chromatogram under the peak.

Parameters **ion** (`float`) – The ion to calculate the area for.

Return type `Optional[float]`

Returns The area of the ion under this peak.

property ion_areas

Returns a copy of the ion areas dict.

Return type `Dict`

Returns The dictionary of `ion: ion area` pairs

make_UID ()

Create a unique peak ID (UID).

The UID comprises the retention time of the peak to two decimal places. Subclasses may define a more unique ID.

Author Andrew Isaac

property rt

The retention time of the peak, in seconds.

Return type `float`

set_bounds (*left*, *apex*, *right*)

Sets peak boundaries in points.

Parameters

- **left** (`int`) – Left peak boundary, in points offset from apex
- **apex** (`int`) – Apex of the peak, in points
- **right** (`int`) – Right peak boundary, in points offset from apex

set_ion_area (*ion*, *area*)

Sets the area for a single ion.

Parameters

- **ion** (`int`) – the ion whose area is being entered.
- **area** (`float`) – the area under the IC of ion.

Author Sean O’Callaghan

class `ICPeak` (`rt=0.0`, `mass=None`, `minutes=False`, `outlier=False`)

Bases: `AbstractPeak`

Subclass of `Peak` representing a peak in an ion chromatogram for a single mass.

Parameters

- **rt** (`Union[int, float]`) – Retention time. Default `0.0`.
- **mass** (`Optional[float]`) – The mass of the ion. Default `None`.
- **minutes** (`bool`) – Retention time units flag. If `True`, retention time is in minutes; if `False` retention time is in seconds. Default `False`.
- **outlier** (`bool`) – Whether the peak is an outlier. Default `False`.

Authors Vladimir Likic, Andrew Isaac, Dominic Davis-Foster (type assertions and properties), David Kainer (outlier flag)

New in version 2.3.0.

Attributes:

<code>UID</code>	Return the unique peak ID (UID), either:
<code>area</code>	The area under the peak.
<code>bounds</code>	The peak boundaries in points.
<code>ic_mass</code>	The mass for a single ion chromatogram peak.
<code>ion_areas</code>	Returns a copy of the ion areas dict.
<code>rt</code>	The retention time of the peak, in seconds.

Methods:

<code>__eq__(other)</code>	Return whether this Peak object is equal to another object.
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>get_ion_area(ion)</code>	Returns the area of a single ion chromatogram under the peak.
<code>make_UID()</code>	Create a unique peak ID (UID):
<code>set_bounds(left, apex, right)</code>	Sets peak boundaries in points.
<code>set_ion_area(ion, area)</code>	Sets the area for a single ion.

property `UID`

Return the unique peak ID (UID), either:

- Integer masses of top two intensities and their ratio (as `Mass1-Mass2-Ratio*100`); or
- the single mass as an integer and the retention time.

Return type `str`

Returns UID string

Author Andrew Isaac

__eq__ (*other*)

Return whether this Peak object is equal to another object.

Parameters *other* – The other object to test equality with.

Return type `bool`

property area

The area under the peak.

Author Andrew Isaac

Return type `Optional[float]`

property bounds

The peak boundaries in points.

Return type `Optional[Tuple[int, int, int]]`

Returns A 3-element tuple containing the left, apex, and right peak boundaries in points. Left and right are offsets.

Author Andrew Isaac

dump (*file_name*, *protocol*=3)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – Filename to save the dump as.
- **protocol** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

get_ion_area (*ion*)

Returns the area of a single ion chromatogram under the peak.

Parameters *ion* (`float`) – The ion to calculate the area for.

Return type `Optional[float]`

Returns The area of the ion under this peak.

property ic_mass

The mass for a single ion chromatogram peak.

Return type `Optional[float]`

Returns The mass of the single ion chromatogram that the peak is from

property ion_areas

Returns a copy of the ion areas dict.

Return type `Dict`

Returns The dictionary of `ion: ion area` pairs

make_UID ()

Create a unique peak ID (UID):

- the single mass as an integer and the retention time.

Author Andrew Isaac

property `rt`

The retention time of the peak, in seconds.

Return type `float`

set_bounds (*left, apex, right*)

Sets peak boundaries in points.

Parameters

- **left** (`int`) – Left peak boundary, in points offset from apex
- **apex** (`int`) – Apex of the peak, in points
- **right** (`int`) – Right peak boundary, in points offset from apex

set_ion_area (*ion, area*)

Sets the area for a single ion.

Parameters

- **ion** (`int`) – the ion whose area is being entered.
- **area** (`float`) – the area under the IC of ion.

Author Sean O’Callaghan

class `Peak` (*rt: float, ms: Optional[pyms.Spectrum.MassSpectrum], minutes: bool = ..., outlier: bool = ...*)

class `Peak` (*rt: float, ms: float, minutes: bool = ..., outlier: bool = ...*)

Bases: `AbstractPeak`

Subclass of `Peak` representing a peak in a mass spectrum.

Parameters

- **rt** (`Union[int, float]`) – Retention time. Default `0.0`.
- **ms** (`Union[float, MassSpectrum, None]`) – The mass spectrum at the apex of the peak. Default `None`.
- **minutes** (`bool`) – Retention time units flag. If `True`, retention time is in minutes; if `False` retention time is in seconds. Default `False`.
- **outlier** (`bool`) – Whether the peak is an outlier. Default `False`.

Authors Vladimir Likic, Andrew Isaac, Dominic Davis-Foster (type assertions and properties), David Kainer (outlier flag)

Changed in version 2.3.0: Functionality related to single ion peaks has moved to the `ICPeak` class. The two classes share a common base class, `AbstractPeak`, which can be used in type checks for functions that accept either type of peak.

Changed in version 2.3.0: If the `ms` argument is unset an empty mass spectrum is used, rather than `None` in previous versions.

Attributes:

<i>UID</i>	Return the unique peak ID (UID), either:
<i>area</i>	The area under the peak.
<i>bounds</i>	The peak boundaries in points.
<i>ion_areas</i>	Returns a copy of the ion areas dict.
<i>mass_spectrum</i>	The mass spectrum at the apex of the peak.
<i>rt</i>	The retention time of the peak, in seconds.

Methods:

<code>__eq__(other)</code>	Return whether this Peak object is equal to another object.
<code>crop_mass(mass_min, mass_max)</code>	Crops mass spectrum.
<code>dump(file_name[, protocol])</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>find_mass_spectrum(data[, from_bounds])</code>	
<code>get_int_of_ion(ion)</code>	Returns the intensity of a given ion in this peak.
<code>get_ion_area(ion)</code>	Returns the area of a single ion chromatogram under the peak.
<code>get_third_highest_mz()</code>	Returns the m/z value with the third highest intensity.
<code>make_UID()</code>	Create a unique peak ID (UID):
<code>null_mass(mass)</code>	Ignore given mass in spectra.
<code>set_bounds(left, apex, right)</code>	Sets peak boundaries in points.
<code>set_ion_area(ion, area)</code>	Sets the area for a single ion.
<code>top_ions([num_ions])</code>	Computes the highest #num_ions intensity ions.

property UID

Return the unique peak ID (UID), either:

- Integer masses of top two intensities and their ratio (as `Mass1-Mass2-Ratio*100`); or
- the single mass as an integer and the retention time.

Return type `str`

Returns UID string

Author Andrew Isaac

`__eq__(other)`

Return whether this Peak object is equal to another object.

Parameters `other` – The other object to test equality with.

Return type `bool`

property area

The area under the peak.

Author Andrew Isaac

Return type `Optional[float]`

property bounds

The peak boundaries in points.

Return type `Optional[Tuple[int, int, int]]`

Returns A 3-element tuple containing the left, apex, and right peak boundaries in points. Left and right are offsets.

Author Andrew Isaac

crop_mass (*mass_min*, *mass_max*)

Crops mass spectrum.

Parameters

- **mass_min** (`float`) – Minimum mass value.
- **mass_max** (`float`) – Maximum mass value.

Author Andrew Isaac

dump (*file_name*, *protocol=3*)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – Filename to save the dump as.
- **protocol** (`int`) – The pickle protocol to use. Default 3.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib and pickle protocol support)

find_mass_spectrum (*data*, *from_bounds=False*)

Sets the peak's mass spectrum from the data.

Clears the single ion chromatogram mass.

Parameters

- **data** (`BaseIntensityMatrix`)
- **from_bounds** (`float`) – Whether to use the attribute `pyms.Peak.Class.Peak.pt_bounds` or to find the peak apex from the peak retention time. Default `False`.

get_int_of_ion (*ion*)

Returns the intensity of a given ion in this peak.

Parameters **ion** (`int`) – The m/z value of the ion of interest

Return type `float`

get_ion_area (*ion*)

Returns the area of a single ion chromatogram under the peak.

Parameters **ion** (`float`) – The ion to calculate the area for.

Return type `Optional[float]`

Returns The area of the ion under this peak.

get_third_highest_mz()

Returns the m/z value with the third highest intensity.

Return type `int`

property ion_areas

Returns a copy of the ion areas dict.

Return type `Dict`

Returns The dictionary of `ion: ion area` pairs

make_UID()

Create a unique peak ID (UID):

- Integer masses of top two intensities and their ratio (as `Mass1-Mass2-Ratio*100`); or

Author Andrew Isaac

property mass_spectrum

The mass spectrum at the apex of the peak.

Return type `MassSpectrum`

null_mass(mass)

Ignore given mass in spectra.

Parameters `mass (float)` – Mass value to remove

Author Andrew Isaac

property rt

The retention time of the peak, in seconds.

Return type `float`

set_bounds(left, apex, right)

Sets peak boundaries in points.

Parameters

- **left (int)** – Left peak boundary, in points offset from apex
- **apex (int)** – Apex of the peak, in points
- **right (int)** – Right peak boundary, in points offset from apex

set_ion_area(ion, area)

Sets the area for a single ion.

Parameters

- **ion (int)** – the ion whose area is being entered.
- **area (float)** – the area under the IC of ion.

Author Sean O’Callaghan

top_ions (*num_ions=5*)

Computes the highest #num_ions intensity ions.

Parameters **num_ions** (*int*) – The number of ions to be recorded. Default 5.

Return type `List[float]`

Returns A list of the ions with the highest intensity.

Authors Sean O’Callaghan, Dominic Davis-Foster (type assertions)

pyms.Peak.Function

Functions related to Peak modification.

Functions:

<i>half_area</i> (ia[, max_bound, tol])	Find bound of peak by summing intensities until change in sum is less than <code>tol</code> percent of the current area.
<i>ion_area</i> (ia, apex[, max_bound, tol])	Find bounds of peak by summing intensities until change in sum is less than <code>tol</code> percent of the current area.
<i>median_bounds</i> (im, peak[, shared])	Calculates the median of the left and right bounds found for each apexing peak mass.
<i>peak_pt_bounds</i> (im, peak)	Approximate the peak bounds (left and right offsets from apex).
<i>peak_sum_area</i> (im, peak[, single_ion, max_bound])	Calculate the sum of the raw ion areas based on detected boundaries.
<i>peak_top_ion_areas</i> (im, peak[, n_top_ions, ...])	Calculate and return the ion areas of the five most abundant ions in the peak.
<i>top_ions_v1</i> (peak[, num_ions])	Computes the highest 5 intensity ions.
<i>top_ions_v2</i> (peak[, num_ions])	Computes the highest #num_ions intensity ions.

half_area (*ia, max_bound=0, tol=0.5*)

Find bound of peak by summing intensities until change in sum is less than `tol` percent of the current area.

Parameters

- **ia** (`List`) – List of intensities from Peak apex for a given mass.
- **max_bound** (`int`) – Optional value to limit size of detected bound. Default 0.
- **tol** (`float`) – Percentage tolerance of added area to current area. Default 0.5.

Return type `Tuple[float, float, float]`

Returns Half peak area, boundary offset, shared (True if shared ion).

Authors Andrew Isaac, Dominic Davis-Foster (type assertions)

ion_area (*ia, apex, max_bound=0, tol=0.5*)

Find bounds of peak by summing intensities until change in sum is less than `tol` percent of the current area.

Parameters

- **ia** (`List`) – List of intensities for a given mass.
- **apex** (`int`) – Index of the peak apex.

- **max_bound** (*int*) – Optional value to limit size of detected bound. Default 0.
- **tol** (*float*) – Percentage tolerance of added area to current area. Default 0.5.

Return type `Tuple[float, float, float, float, float]`

Returns Area, left and right boundary offset, shared left, shared right.

Authors Andrew Isaac, Dominic Davis-Foster (type assertions)

median_bounds (*im, peak, shared=True*)

Calculates the median of the left and right bounds found for each apexing peak mass.

Parameters

- **im** (*BaseIntensityMatrix*) – The originating IntensityMatrix object.
- **peak** (*Peak*)
- **shared** (*bool*) – Include shared ions shared with neighbouring peak. Default `True`.

Return type `Tuple[float, float]`

Returns Median left and right boundary offset in points.

Authors Andrew Isaac, Dominic Davis-Foster

peak_pt_bounds (*im, peak*)

Approximate the peak bounds (left and right offsets from apex).

Parameters

- **im** (*BaseIntensityMatrix*) – The originating IntensityMatrix object
- **peak** (*Peak*)

Return type `Tuple[int, int]`

Returns Sum of peak apex ions in detected bounds

Authors Andrew Isaac, Sean O’Callaghan, Dominic Davis-Foster

peak_sum_area (*im, peak, single_ion=False, max_bound=0*)

Calculate the sum of the raw ion areas based on detected boundaries.

Parameters

- **im** (*BaseIntensityMatrix*) – The originating IntensityMatrix object.
- **peak** (*Peak*)
- **single_ion** (*bool*) – whether single ion areas should be returned. Default `False`.
- **max_bound** (*int*) – Optional value to limit size of detected bound. Default 0.

Return type `Union[float, Tuple[float, Dict[float, float]]]`

Returns Sum of peak apex ions in detected bounds.

Overloads

- `peak_sum_area(im, peak, single_ion: Literal[True], max_bound = ...) -> Tuple[float, Dict[float, float]]`
- `peak_sum_area(im, peak, single_ion: Literal[False] = ..., max_bound = ...) -> float`

Authors Andrew Isaac, Dominic Davis-Foster (type assertions)

peak_top_ion_areas (*im, peak, n_top_ions=5, max_bound=0*)

Calculate and return the ion areas of the five most abundant ions in the peak.

Parameters

- **im** (*IntensityMatrix*) – The originating IntensityMatrix object.
- **peak** (*Peak*)
- **n_top_ions** (*int*) – Number of top ions to return areas for. Default 5.
- **max_bound** (*int*) – Optional value to limit size of detected bound. Default 0.

Return type `Dict[float, float]`

Returns Dictionary of `ion : ion_area` pairs.

Authors Sean O’Callaghan, Dominic Davis-Foster (type assertions)

top_ions_v1 (*peak, num_ions=5*)

Computes the highest 5 intensity ions.

Parameters

- **peak** (*Peak*) – the peak to be processed.
- **num_ions** (*int*) – The number of ions to be recorded. Default 5.

Return type `List[float]`

Returns A list of the top 5 highest intensity ions

Authors Sean O’Callaghan, Dominic Davis-Foster (type assertions)

Deprecated since version 2.0.0: This will be removed in 2.4.0. Use `pyms.Peak.Function.top_ions_v2()` instead

top_ions_v2 (*peak, num_ions=5*)

Computes the highest #num_ions intensity ions.

Parameters

- **peak** (*Peak*) – The peak to be processed
- **num_ions** (*int*) – The number of ions to be recorded. Default 5.

Return type `List[float]`

Returns A list of the num_ions highest intensity ions

Authors Sean O’Callaghan, Dominic Davis-Foster (type assertions)

Deprecated since version 2.1.2: This will be removed in 2.5.0. Use `pyms.Peak.Class.Peak.top_ions()` instead

pyms.Peak.List

Functions for modelling peak lists.

pyms.Peak.List.Function

Functions related to Peak modification.

Functions:

<code>composite_peak(peak_list[, ignore_outliers])</code>	Create a peak that consists of a composite spectrum from all spectra in the list of peaks.
<code>fill_peaks(data, peak_list, D[, minutes])</code>	Gets the best matching Retention Time and spectra from 'data' for each peak in the peak list.
<code>is_peak_list(peaks)</code>	Returns whether peaks is a valid peak list.
<code>sele_peaks_by_rt(peaks, rt_range)</code>	Selects peaks from a retention time range.

composite_peak (*peak_list*, *ignore_outliers=False*)

Create a peak that consists of a composite spectrum from all spectra in the list of peaks.

Parameters

- **peak_list** (*List[Peak]*) – A list of peak objects
- **ignore_outliers** (*bool*) – Default `False`.

Return type *Optional[Peak]*

Returns The composite peak

Authors Andrew Isaac, Dominic Davis-Foster (type assertions)

fill_peaks (*data*, *peak_list*, *D*, *minutes=False*)

Gets the best matching Retention Time and spectra from 'data' for each peak in the peak list.

Parameters

- **data** (*BaseIntensityMatrix*) – A data IntensityMatrix that has the same mass range as the peaks in the peak list
- **peak_list** (*List[Peak]*) – A list of peak objects
- **D** (*float*) – Peak width standard deviation in seconds. Determines search window width.
- **minutes** (*bool*) – Return retention time as minutes. Default `False`.

Return type *List[Peak]*

Returns List of Peak Objects

Authors Andrew Isaac, Dominic Davis-Foster (type assertions)

is_peak_list (*peaks*)

Returns whether peaks is a valid peak list.

Author Dominic Davis-Foster

Return type *bool*

select_peaks_by_rt (*peaks*, *rt_range*)

Selects peaks from a retention time range.

Parameters

- **peaks** (`Union[Sequence[Peak], ndarray]`) – A list of peak objects
- **rt_range** (`Sequence[str]`) – A list of two time strings, specifying lower and upper retention times.

Return type `List[Peak]`

Returns A list of peak objects

`pym.s.Peak.List.IO`

Functions related to storing and loading a list of Peak objects.

Functions:

<code>load_peaks(file_name)</code>	Loads the peak_list stored with <code>store_peaks()</code> .
<code>store_peaks(peak_list, file_name[, protocol])</code>	Store the list of peak objects.

load_peaks (*file_name*)

Loads the peak_list stored with `store_peaks()`.

Parameters **file_name** (`Union[str, Path, PathLike]`) – File name of peak list

Return type `List[Peak]`

Returns The list of Peak objects

Authors Andrew Isaac, Dominic Davis-Foster (pathlib support)

store_peaks (*peak_list*, *file_name*, *protocol=1*)

Store the list of peak objects.

Parameters

- **peak_list** (`Sequence[Peak]`) – A list of peak objects.
- **file_name** (`Union[str, Path, PathLike]`) – File name to store peak list.
- **protocol** (`int`) – The `pickle` protocol to use. Default 1.

Authors Andrew Isaac, Dominic Davis-Foster (type assertions and pathlib support)

6.3.15 `pym.s.Simulator`

Table of Contents

- `pym.s.Simulator`

Provides functions for simulation of GCMS data.

Functions:

<code>add_gaussc_noise(im, scale)</code>	Adds noise to an IntensityMatrix object.
<code>add_gaussc_noise_ic(ic, scale)</code>	Adds noise drawn from a normal distribution with constant scale to an ion chromatogram.
<code>add_gaussv_noise(im, scale, cutoff, prop)</code>	Adds noise to an IntensityMatrix object.
<code>add_gaussv_noise_ic(ic, scale, cutoff, prop)</code>	Adds noise to an ic.
<code>chromatogram(n_scan, x_zero, sigma, peak_scale)</code>	Returns a simulated ion chromatogram of a pure component.
<code>gaussian(point, mean, sigma, scale)</code>	Calculates a point on a gaussian density function.
<code>gcms_sim(time_list, mass_list, peak_list)</code>	Simulator of GCMS data.

add_gaussc_noise (*im, scale*)

Adds noise to an IntensityMatrix object.

Parameters

- **im** (*BaseIntensityMatrix*) – the intensity matrix object
- **scale** (*float*) – the scale of the normal distribution from which the noise is drawn

Author Sean O’Callaghan

add_gaussc_noise_ic (*ic, scale*)

Adds noise drawn from a normal distribution with constant scale to an ion chromatogram.

Parameters

- **ic** (*IonChromatogram*) – The ion Chromatogram.
- **scale** (*float*) – The scale of the normal distribution.

Author Sean O’Callaghan

add_gaussv_noise (*im, scale, cutoff, prop*)

Adds noise to an IntensityMatrix object.

Parameters

- **im** (*BaseIntensityMatrix*) – the intensity matrix object
- **scale** (*int*) – the scale of the normal distribution from which the noise is drawn
- **cutoff** (*int*) – The level below which the intensity of the ic at that point has no effect on the scale of the noise distribution
- **scale** – The scale of the normal distribution for ic values
- **prop** (*float*) – For intensity values above the cutoff, the scale is multiplied by the ic value multiplied by prop.

Author Sean O’Callaghan

add_gaussv_noise_ic (*ic, scale, cutoff, prop*)

Adds noise to an ic. The noise value is drawn from a normal distribution, the scale of this distribution depends on the value of the ic at the point where the noise is being added

Parameters

- **ic** (*IonChromatogram*) – The IonChromatogram

- **cutoff** (*int*) – The level below which the intensity of the ic at that point has no effect on the scale of the noise distribution
- **scale** (*int*) – The scale of the normal distribution for ic values below the cutoff is modified for values above the cutoff
- **prop** (*float*) – For ic values above the cutoff, the scale is multiplied by the ic value multiplied by prop.

Author Sean O’Callaghan

chromatogram (*n_scan, x_zero, sigma, peak_scale*)

Returns a simulated ion chromatogram of a pure component.

The ion chromatogram contains a single gaussian peak.

Parameters

- **n_scan** (*int*) – the number of scans
- **x_zero** (*int*) – The apex of the peak
- **sigma** (*float*) – The standard deviation of the distribution
- **peak_scale** (*float*) – the intensity of the peak at the apex

Author Sean O’Callaghan

Return type *ndarray*

gaussian (*point, mean, sigma, scale*)

Calculates a point on a gaussian density function.

$$f = s * \exp(-(x - x_0)^2 / (2 * w^2))$$

Parameters

- **point** (*float*) – The point currently being computed
- **mean** (*int*) – The apex of the peak
- **sigma** (*float*) – The standard deviation of the gaussian
- **scale** (*float*) – The height of the apex

Return type *float*

Returns a single value from a normal distribution

Author Sean O’Callaghan

gcms_sim (*time_list, mass_list, peak_list*)

Simulator of GCMS data.

Parameters

- **time_list** (*List[float]*) – the list of scan times
- **mass_list** (*List[float]*) – the list of m/z channels
- **peak_list** (*List[Peak]*) – A list of peaks

Return type *IntensityMatrix*

Returns A simulated Intensity Matrix object

Author Sean O’Callaghan

6.3.16 `pyms.TopHat`

Top-hat baseline corrector.

Functions:

<code>tophat(ic[, struct])</code>	Top-hat baseline correction on Ion Chromatogram.
<code>tophat_im(im[, struct])</code>	Top-hat baseline correction on Intensity Matrix.

`tophat` (*ic*, *struct=None*)

Top-hat baseline correction on Ion Chromatogram.

Parameters

- **`ic`** (*IonChromatogram*) – The input ion chromatogram.
- **`struct`** (*Union[int, str, None]*) – Top-hat structural element as time string. The structural element needs to be larger than the features one wants to retain in the spectrum after the top-hat transform. Default *None*.

Return type *IonChromatogram*

Returns Top-hat corrected ion chromatogram.

Authors Woon Wai Keen, Vladimir Likic, Dominic Davis-Foster (type assertions)

`tophat_im` (*im*, *struct=None*)

Top-hat baseline correction on Intensity Matrix.

Wraps around the TopHat function above.

Parameters

- **`im`** (*BaseIntensityMatrix*) – The input Intensity Matrix.
- **`struct`** (*Optional[str]*) – Top-hat structural element as time string. The structural element needs to be larger than the features one wants to retain in the spectrum after the top-hat transform. Default *None*.

Return type *BaseIntensityMatrix*

Returns Top-hat corrected IntensityMatrix Matrix

Author Sean O’Callaghan

6.3.17 `pyms.Utills`

Table of Contents

- `pyms.Utills`
 - `pyms.Utills.IO`
 - `pyms.Utills.Math`

```

- pyms.Utills.Time
- pyms.Utills.Utills

```

Utility functions for PyMassSpec wide use.

`pyms.Utills.IO`

General I/O functions.

Functions:

<code>dump_object(obj, file_name)</code>	Dumps an object to a file through <code>pickle.dump()</code> .
<code>file_lines(file_name[, strip])</code>	Returns lines from a file, as a list.
<code>load_object(file_name)</code>	Loads an object previously dumped with <code>dump_object()</code> .
<code>prepare_filepath(file_name[, makedirs])</code>	Convert string filename into <code>pathlib.Path</code> object and create parent directories if required.
<code>save_data(file_name, data[, format_str, ...])</code>	Saves a list of numbers or a list of lists of numbers to a file with specific formatting.

`dump_object` (*obj, file_name*)

Dumps an object to a file through `pickle.dump()`.

Parameters

- **`obj`** (*Any*) – Object to be dumped
- **`file_name`** (*Union[str, Path, PathLike]*) – Name of the file for the object dump

Authors Vladimir Likic, Dominic Davis-Foster (pathlib support)

`file_lines` (*file_name, strip=False*)

Returns lines from a file, as a list.

Parameters

- **`file_name`** (*Union[str, Path, PathLike]*) – Name of a file
- **`strip`** (*bool*) – If True, lines are pre-processed. Newline characters are removed, leading and trailing whitespaces are removed, and lines starting with '#' are discarded Default `False`.

Return type `List[str]`

Returns A list of lines

Authors Vladimir Likic, Dominic Davis-Foster (pathlib support)

`load_object` (*file_name*)

Loads an object previously dumped with `dump_object()`.

Parameters **`file_name`** (*Union[str, Path, PathLike]*) – Name of the object dump file.

Return type `object`

Returns Object contained in the file.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib support)

prepare_filepath (*file_name*, *mkdirs=True*)

Convert string filename into pathlib.Path object and create parent directories if required.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – file_name to process
- **mkdirs** (`bool`) – Whether the parent directory of the file should be created if it doesn't exist. Default `True`.

Return type `Path`

Returns file_name

Author Dominic Davis-Foster

save_data (*file_name*, *data*, *format_str='%f'*, *prepend=""*, *sep=' '*, *compressed=False*)

Saves a list of numbers or a list of lists of numbers to a file with specific formatting.

Parameters

- **file_name** (`Union[str, Path, PathLike]`) – Name of a file
- **data** (`Union[List[float], List[List[float]]]`) – A list of numbers, or a list of lists
- **format_str** (`str`) – A format string for individual entries. Default `'%.6f'`.
- **prepend** (`str`) – A string, printed before each row. Default `' '`.
- **sep** (`str`) – A string, printed after each number. Default `'_'`.
- **compressed** (`bool`) – If `True`, the output will be gzipped. Default `False`.

Authors Vladimir Likic, Dominic Davis-Foster (pathlib support)

pym.s.Utils.Math

Provides mathematical functions.

Functions:

<code>MAD(v)</code>	Median absolute deviation.
<code>is_float(s)</code>	Test if a string, or list of strings, contains a numeric value(s).
<code>mad_based_outlier(data[, thresh])</code>	Identify outliers using the median absolute deviation (MAD).
<code>mean(data)</code>	Return the sample arithmetic mean of data.
<code>median(data)</code>	Return the median (middle value) of numeric data.
<code>median_outliers(data[, m])</code>	Identify outliers using the median value.
<code>percentile_based_outlier(data[, threshold])</code>	Identify outliers using a percentile.
<code>rmsd(list1, list2)</code>	Calculates RMSD for the 2 lists.
<code>std(data[, xbar])</code>	Return the square root of the sample variance.
<code>vector_by_step(start, stop, step)</code>	Generates a list by using start, stop, and step values.

MAD (*v*)

Median absolute deviation.

Parameters **v** (`Union[Sequence, ndarray]`) – List of values to calculate the median absolute deviation of.

Return type `float`

Returns median absolute deviation

Author Vladimir Likic

is_float (*s*)

Test if a string, or list of strings, contains a numeric value(s).

Parameters *s* (`Union[str, List[str]]`) – The string or list of strings to test.

Return type `Union[bool, List[bool]]`

Returns A single boolean or list of boolean values indicating whether each input can be converted into a float.

Overloads

- `is_float(s: str) -> bool`
- `is_float(s: List[str]) -> List[bool]`

mad_based_outlier (*data, thresh=3.5*)

Identify outliers using the median absolute deviation (MAD).

Parameters

- **data**
- **thresh** (`float`) – Default 3.5.

Author David Kainer

Url <http://stackoverflow.com/questions/22354094/pythonic-way-of-detecting-outliers-in-one-dimensional-observation-data>

mean (*data*)

Return the sample arithmetic mean of data.

```
>>> mean([1, 2, 3, 4, 4])
2.8
```

```
>>> from fractions import Fraction as F
>>> mean([F(3, 7), F(1, 21), F(5, 3), F(1, 3)])
Fraction(13, 21)
```

```
>>> from decimal import Decimal as D
>>> mean([D("0.5"), D("0.75"), D("0.625"), D("0.375")])
Decimal('0.5625')
```

If data is empty, `StatisticsError` will be raised.

median (*data*)

Return the median (middle value) of numeric data.

When the number of data points is odd, return the middle data point. When the number of data points is even, the median is interpolated by taking the average of the two middle values:

```
>>> median([1, 3, 5])
3
```

(continues on next page)

(continued from previous page)

```
>>> median([1, 3, 5, 7])
4.0
```

median_outliers (*data*, *m*=2.5)

Identify outliers using the median value.

Parameters

- **data**
- **m** (*float*) – Default 2.5.

Author David Kainer**Author** eumiro (<https://stackoverflow.com/users/449449/eumiro>)**Author** Benjamin Bannier(<https://stackoverflow.com/users/176922/benjamin-bannier>)**Url** <http://stackoverflow.com/questions/11686720/is-there-a-numpy-builtin-to-reject-outliers-from-a-list>**percentile_based_outlier** (*data*, *threshold*=95)

Identify outliers using a percentile.

Parameters

- **data**
- **threshold** (*int*) – Default 95.

Author David Kainer**Url** <http://stackoverflow.com/questions/22354094/pythonic-way-of-detecting-outliers-in-one-dimensional-observation-data>**rmsd** (*list1*, *list2*)

Calculates RMSD for the 2 lists.

Parameters

- **list1** (*Union[Sequence, ndarray]*) – First data set
- **list2** (*Union[Sequence, ndarray]*) – Second data set

Return type *float***Returns** RMSD value**Authors** Qiao Wang, Andrew Isaac, Vladimir Likic**std** (*data*, *xbar*=None)

Return the square root of the sample variance.

See variance for arguments and other details.

```
>>> stdev([1.5, 2.5, 2.5, 2.75, 3.25, 4.75])
1.0810874155219827
```


vector_by_step (*start, stop, step*)

Generates a list by using start, stop, and step values.

Parameters

- **start** (*float*) – Initial value
- **stop** (*float*) – Max value
- **step** (*float*) – Step

Author Vladimir Likic

Return type `List[float]`

pym.s.Utils.Time

Time conversion and related functions.

Functions:

<code>is_str_num(arg)</code>	Returns whether the argument is a string in the format of a number.
<code>time_str_secs(time_str)</code>	Resolves time string of the form ' <code><NUMBER>s</code> ' or ' <code><NUMBER>m</code> ' and returns the time in seconds.
<code>window_sele_points(ic, window_sele[, ...])</code>	Converts window selection parameter into points based on the time step in an ion chromatogram

is_str_num (*arg*)

Returns whether the argument is a string in the format of a number.

The number can be an integer, or alternatively a floating point number in scientific or engineering format.

Parameters **arg** (*str*) – A string to be evaluate as a number

Author Gyro Funch (from Active State Python Cookbook)

Return type `bool`

time_str_secs (*time_str*)

Resolves time string of the form '`<NUMBER>s`' or '`<NUMBER>m`' and returns the time in seconds.

Parameters **time_str** (*str*) – A time string, which must be of the form '`<NUMBER>s`' or '`<NUMBER>m`' where '`<NUMBER>`' is a valid number

Return type `float`

Returns Time in seconds

Author Vladimir Likic

window_sele_points (*ic, window_sele, half_window=False*)

Converts window selection parameter into points based on the time step in an ion chromatogram

Parameters

- **ic** (*IonChromatogram*) – ion chromatogram object relevant for the conversion
- **window_sele** (*Union[int, str]*) – The window selection parameter. This can be an integer or

time string. If an integer, taken as the number of points. If a string, must of the form '`<NUMBER>s`' or '`<NUMBER>m`', specifying a time in seconds or minutes, respectively

- **half_window** (`bool`) – Specifies whether to return half-window. Default `False`.

Return type `int`

Returns The number of points in the window

Author Vladimir Likic

`pyms.Utills.Utills`

General utility functions.

Functions:

<code>is_number(obj)</code>	Returns whether <code>obj</code> is a numerical value (<code>int</code> , <code>:class`float`</code> etc).
<code>is_path(obj)</code>	Returns whether the object represents a filesystem path.
<code>is_sequence(obj)</code>	Returns whether the object is a <code>Sequence</code> , and not a string.
<code>is_sequence_of(obj, of)</code>	Returns whether the object is a <code>Sequence</code> , and not a string, of the given type.

is_number (`obj`)

Returns whether `obj` is a numerical value (`int`, `:class`float`` etc).

Parameters `obj` (`Any`)

Return type `bool`

is_path (`obj`)

Returns whether the object represents a filesystem path.

Parameters `obj` (`Any`)

Return type `bool`

is_sequence (`obj`)

Returns whether the object is a `Sequence`, and not a string.

Parameters `obj` (`Any`)

Return type `bool`

is_sequence_of (`obj, of`)

Returns whether the object is a `Sequence`, and not a string, of the given type.

Parameters

- `obj` (`Any`)
- `of` (`Any`)

Return type `bool`

`pyms.Utills.Utills.signedinteger`
`numpy.signedinteger` at runtime; `int` when type checking.

6.3.18 Changelog

Changes in v2.3.0

- All functions, classes and methods now have **PEP 484** type hints. Contributed by Chris Davis-Foster in #4
- All modules now implement `__all__` to limit the objects imported when using `*` imports.
- Removed the following deprecated functions:

Removed object	Suggested replacement
<code>pyms.Experiment.Experiment.get_expr_code()</code>	<code>pyms.Experiment.Experiment.expr</code>
<code>pyms.Experiment.Experiment.get_peak_list()</code>	<code>pyms.Experiment.Experiment.peaks</code>
<code>pyms.Experiment.Experiment.store()</code>	<code>pyms.Experiment.Experiment.dump()</code>
<code>pyms.Experiment.store_expr()</code>	<code>pyms.Experiment.Experiment.dump_expr()</code>
<code>pyms.GCMS.Class.GCMS_data.get_scan_list()</code>	<code>pyms.GCMS.Class.GCMS_data.scans</code>
<code>pyms.GCMS.Class.GCMS_data.get_tic()</code>	<code>pyms.GCMS.Class.GCMS_data.tic</code>
<code>pyms.Gapfill.Class.MissingPeak.get_common_ion()</code>	<code>pyms.Gapfill.Class.MissingPeak.common_ion</code>
<code>pyms.Gapfill.Class.MissingPeak.get_common_ion_area()</code>	<code>pyms.Gapfill.Class.MissingPeak.common_ion_area</code>
<code>pyms.Gapfill.Class.MissingPeak.get_exact_rt()</code>	<code>pyms.Gapfill.Class.MissingPeak.exact_rt</code>
<code>pyms.Gapfill.Class.MissingPeak.get_qual_ion1()</code>	<code>pyms.Gapfill.Class.MissingPeak.qual_ion1</code>
<code>pyms.Gapfill.Class.MissingPeak.get_qual_ion2()</code>	<code>pyms.Gapfill.Class.MissingPeak.qual_ion2</code>
<code>pyms.Gapfill.Class.MissingPeak.get_rt()</code>	<code>pyms.Gapfill.Class.MissingPeak.rt</code>
<code>pyms.Gapfill.Class.MissingPeak.set_common_ion_area()</code>	<code>pyms.Gapfill.Class.MissingPeak.common_ion_area</code>
<code>pyms.Gapfill.Class.MissingPeak.set_exact_rt()</code>	<code>pyms.Gapfill.Class.MissingPeak.exact_rt</code>
<code>pyms.Gapfill.Class.Sample.get_missing_peaks()</code>	<code>pyms.Gapfill.Class.Sample.missing_peaks</code>
<code>pyms.Gapfill.Class.Sample.get_mp_rt_area_dict()</code>	<code>pyms.Gapfill.Class.Sample.rt_area_dict</code>
<code>pyms.Gapfill.Class.Sample.get_name()</code>	<code>pyms.Gapfill.Class.Sample.name</code>
<code>pyms.Gapfill.Function.transposed()</code>	
<code>pyms.IonChromatogram.IonChromatogram.get_mass()</code>	<code>pyms.IonChromatogram.IonChromatogram.mass</code>
<code>pyms.IonChromatogram.IonChromatogram.get_time_step()</code>	<code>pyms.IonChromatogram.IonChromatogram.time_step</code>
<code>pyms.IonChromatogram.IonChromatogram.set_intensity_array()</code>	<code>pyms.IonChromatogram.IonChromatogram.intensity_array</code>
<code>pyms.Mixins.MaxMinMassMixin.get_max_mass()</code>	<code>pyms.Mixins.MaxMinMassMixin.max_mass</code>
<code>pyms.Mixins.MaxMinMassMixin.get_min_mass()</code>	<code>pyms.Mixins.MaxMinMassMixin.min_mass</code>
<code>pyms.Mixins.MassListMixin.get_mass_list()</code>	<code>pyms.Mixins.MassListMixin.mass_list</code>
<code>pyms.Mixins.TimeListMixin.get_time_list()</code>	<code>pyms.Mixins.TimeListMixin.time_list</code>
<code>pyms.Mixins.IntensityArrayMixin.get_intensity_array()</code>	<code>pyms.Mixins.IntensityArrayMixin.intensity_array</code>
<code>pyms.Mixins.IntensityArrayMixin.get_matrix_list()</code>	<code>pyms.Mixins.IntensityArrayMixin.matrix_list</code>
<code>pyms.Peak.Class.Peak.get_area()</code>	<code>pyms.Peak.Class.Peak.area</code>
<code>pyms.Peak.Class.Peak.get_ic_mass()</code>	<code>pyms.Peak.Class.ICPeak.ic_mass</code>
<code>pyms.Peak.Class.Peak.get_ion_areas()</code>	<code>pyms.Peak.Class.Peak.ion_areas</code>
<code>pyms.Peak.Class.Peak.get_mass_spectrum()</code>	<code>pyms.Peak.Class.Peak.mass_spectrum</code>
<code>pyms.Peak.Class.Peak.get_pt_bounds()</code>	<code>pyms.Peak.Class.Peak.bounds</code>
<code>pyms.Peak.Class.Peak.get_rt()</code>	<code>pyms.Peak.Class.Peak.rt</code>
<code>pyms.Peak.Class.Peak.get_UID()</code>	<code>pyms.Peak.Class.Peak.UID</code>
<code>pyms.Peak.Class.Peak.set_area()</code>	<code>pyms.Peak.Class.Peak.area</code>
<code>pyms.Peak.Class.Peak.set_ic_mass()</code>	<code>pyms.Peak.Class.ICPeak.ic_mass</code>
<code>pyms.Peak.Class.Peak.set_ion_areas()</code>	<code>pyms.Peak.Class.Peak.ion_areas</code>

Table 83 – continued from previous page

Removed object	Suggested replacement
<code>pyms.Peak.Class.Peak.set_mass_spectrum()</code>	<code>pyms.Peak.Class.Peak.mass_spe</code>
<code>pyms.Peak.Class.Peak.set_pt_bounds()</code>	<code>pyms.Peak.Class.Peak.pt_bound</code>
<code>pyms.Utills.Utills.is_positive_int()</code>	
<code>pyms.Utills.Utills.is_list_of_dec_nums()</code>	

- Renamed `pyms.Gapfill.Function.file2matrix()` to `pyms.Gapfill.Function.file2dataframe()`. The function now returns a Pandas DataFrame.
- Split `pyms.IntensityMatrix.IntensityMatrix` into two classes: `pyms.IntensityMatrix.BaseIntensityMatrix` and `pyms.IntensityMatrix.IntensityMatrix`. This makes subclassing easier.
- Split `pyms.Peak.Class.Peak` into three classes: `pyms.Peak.Class.AbstractPeak`, `pyms.Peak.Class.Peak`, `pyms.Peak.Class.ICPeak`. `ICPeak` is returned when a mass is passed to the `Peak` constructor instead of a mass spectrum.
- Added the following functions and classes:

<code>pyms.Gapfill.Function.MissingPeakFiletype(value)</code>	Flag to indicate the filetype for <code>pyms.Gapfill.Function.missing_peak_finder()</code> .
<code>pyms.IntensityMatrix.AsciiFiletypes(value)</code>	Enumeration of supported ASCII filetypes for <code>export_ascii()</code> .
<code>pyms.IntensityMatrix.IntensityMatrix.bpc</code>	Constructs a Base Peak Chromatogram from the data.
<code>pyms.IonChromatogram.IonChromatogram.is_eic()</code>	Returns whether the ion chromatogram is an extracted ion chromatogram (EIC).
<code>pyms.IonChromatogram.IonChromatogram.is_bpc()</code>	Returns whether the ion chromatogram is a base peak chromatogram (BPC).
<code>pyms.IonChromatogram.ExtractedIonChromatogram(...)</code>	Models an extracted ion chromatogram (EIC).
<code>pyms.IonChromatogram.BasePeakChromatogram(...)</code>	Models a base peak chromatogram (BPC).
<code>pyms.Spectrum.array_as_numeric(array)</code>	Convert the given numpy array to a numeric data type.
<code>pyms.Utills.Utills.is_path(obj)</code>	Returns whether the object represents a filesystem path.
<code>pyms.Utills.Utills.is_sequence(obj)</code>	Returns whether the object is a <code>Sequence</code> , and not a string.
<code>pyms.Utills.Utills.is_sequence_of(obj, of)</code>	Returns whether the object is a <code>Sequence</code> , and not a string, of the given type.
<code>pyms.Utills.Utills.is_number(obj)</code>	Returns whether <code>obj</code> is a numerical value (<code>int</code> , <code>float</code> etc).
<code>pyms.eic</code>	Class to model a subset of data from an Intensity Matrix.

- The `ia` parameter of `pyms.IonChromatogram.IonChromatogram` was renamed to `intensity_list`.

Changes in v2.2.22-beta2

- `pym.spectrum.Scan` and `pym.spectrum.MassSpectrum` can now accept any values for mass and intensity that can be converted to a `float` or `int`. This includes strings representing numbers. Previously only `int` and `float` values were permitted.
- If the mass and intensity values supplied to a `pym.spectrum.Scan` or a `pym.spectrum.MassSpectrum` are `float`, `int`, or a data type derived from `numpy.number`, the data is stored in that type. For other data types, such as strings, `decimal.Decimal` etc., the data is stored as `float`.
If the data contains values in mixed types then, in most cases, all values will be converted to `float`. If you wish to control this behaviour you should construct a `numpy.ndarray` with the desired type. See <https://numpy.org/devdocs/user/basics.types.html> for a list of types.
- A `TypeError` is no longer raised when creating a `pym.spectrum.Scan` or a `pym.spectrum.MassSpectrum` with a `float`, `int` etc. rather than a sequence. Instead, value is treated as being the sole element in a list.
- Passing a non-numeric string or a list of non-numeric strings to `pym.spectrum.Scan` or `pym.spectrum.MassSpectrum` now raises a `ValueError` and not a `TypeError` as in previous versions.
- `pym.Peak.Class.Peak.ion_areas()` now accepts dictionary keys as `float` as well as `int`.

Changes in v2.2.22-beta1

- `ANDI_reader()` and `pym.spectrum.Scan` were modified to allow ANDI-MS files to be read if the data either:
 - had the m/z data stored from highest m/z to lowest; or
 - contained 0-length scans.

6.4 Demos and Examples

6.4.1 Introduction

Examples of PyMassSpec use given in the User Guide.

Further examples can be found at <https://github.com/domdfcoding/PyMassSpec/tree/master/pym-demo/jupyter>

Chapter 1 – GC-MS Raw Data Model

20e – Reading of GC-MS raw data in the mzML format, and some properties of raw data object. There isn't yet an example mzML file so this demo doesn't yet do anything.

Chapter 2 – GC-MS data derived objects

32 – Saving IonChromatogram and IntensityMatrix information.

Chapter 5 – Peak alignment by dynamic programming.

64 – Peak alignment with the “common ion” filtering.

6.4.2 PyMassSpec test and example data files

The example data files can be downloaded using the links below:

- GC01_0812_066.tar.gz
- gc01_0812_066.cdf – GC-MS data acquired on Agilent 5975C MSD interfaced with Agilent 7890A GC. Data was exported as NetCDF from Agilent ChemStation.
- gc01_0812_066.jdx – GC-MS data acquired on Agilent 5975C MSD interfaced with Agilent 7890A GC. Data was read with GCMS FileTranslatorPro (Scientific Instrument Services, Inc), and exported in JCAMP-DX format
- a0806_077.cdf
- a0806_078.cdf
- a0806_079.cdf – GC-MS data of a life-cycle stage of a parasite. Each data file is the output of a separate GC-MS processing run of a sample prepared from the same life-cycle stage.
- a0806_140.cdf
- a0806_141.cdf
- a0806_142.cdf – GC-MS data of a different life-cycle stage of the parasite. Each data file is the output of a separate GC-MS processing run of a sample prepared from the same life-cycle stage, but a different stage to the previous samples.
- MM-10.0_1_no_processing.cdf – GC-TOF data acquired on Leco Pegasus machine using ChromaTOF software.
- nist08_test.jca – NIST formatted test data

Copyright © 2006-2011 Vladimir Likic, Bio21 Molecular Science and Biotechnology Institute, the University of Melbourne, Melbourne, Australia. All rights reserved.

6.4.3 20e

Download Source

```

1  """proc.py
2  """
3
4  # TODO: mzML demo; need example mzML file
5
6  import pathlib
7  data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
8  # Change this if the data files are stored in a different location
9
10 from pyms.GCMS.IO.MZML import mzML_reader
11
12
13 # read the raw data
14 mzml_file = data_directory / ".mzML"
15 data = mzML_reader(mzml_file)
16 print(data)
17
18 # raw data operations
19 print("minimum mass found in all data: ", data.min_mass)
20 print("maximum mass found in all data: ", data.max_mass)
21
22 # time
23 time = data.time_list
24 print(time)
25 print("number of retention times: ", len(time))
26 print("retention time of 1st scan: ", time[0], "sec")
27 print("index of 400sec in time_list: ", data.get_index_at_time(400.0))
28
29 # TIC
30 tic = data.tic
31 print(tic)
32 print("number of scans in TIC: ", len(tic))
33 print("start time of TIC: ", tic.get_time_at_index(0), "sec")
34
35 # raw scans
36 scans = data.scan_list
37 print(scans)
38 print(scans[0].mass_list)
39 print("1st mass value for 1st scan: ", scans[0].mass_list[0])
40 print("1st intensity value for 1st scan: ", scans[0].intensity_list[0])
41
42 print("minimum mass found in 1st scan: ", scans[0].min_mass)
43 print("maximum mass found in 1st scan: ", scans[0].max_mass)

```

6.4.4 32

Saving IonChromatogram and IntensityMatrix information.

[Download Source](#)

```
1  """proc.py
2  """
3  from pyms.GCMS.IO.JCAMP import JCAMP_reader
4  from pyms.IntensityMatrix import build_intensity_matrix
5  from pyms.Utills.IO import save_data
6
7  # read the raw data as a GCMS_data object
8  jcamp_file = "data/gc01_0812_066.jdx"
9  data = JCAMP_reader(jcamp_file)
10
11  # IntensityMatrix
12  # must build intensity matrix before accessing any intensity matrix methods.
13
14  # default, float masses with interval (bin interval) of one from min mass
15  print("default intensity matrix, bin interval = 1, boundary +/- 0.5")
16  im = build_intensity_matrix(data)
17
18  #
19  # Saving data
20  #
21
22  # save the intensity matrix values to a file
23  mat = im.intensity_array
24  print("saving intensity matrix intensity values...")
25  save_data("output/im.dat", mat)
26
27  # Export the entire IntensityMatrix as CSV. This will create
28  # data.im.csv, data.mz.csv, and data.rt.csv where
29  # these are the intensity matrix, retention time
30  # vector, and m/z vector in the CSV format
31  print("exporting intensity matrix data...")
32  im.export_ascii("output/data")
33
34  # Export the entire IntensityMatrix as LECO CSV. This is
35  # useful for import into AnalyzerPro
36  print("exporting intensity matrix data to LECO CSV format...")
37  im.export_leco_csv("output/data_leco.csv")
38
39  #
40  # Import saved data
41  #
42
43  from pyms.IntensityMatrix import import_leco_csv
44
45  # import LECO CSV file
46  print("importing intensity matrix data from LECO CSV format...")
47  iim = import_leco_csv("output/data_leco.csv")
48
49  # Check size to original
50  print("Output dimensions:", im.size, " Input dimensions:", iim.size)
```


6.4.5 55

Download Source

```

1  """proc.py
2  """
3
4  from pyms.GCMS.IO.ANDI import ANDI_reader
5  from pyms.IntensityMatrix import build_intensity_matrix_i
6  from pyms.Noise.SavitzkyGolay import savitzky_golay
7  from pyms.TopHat import tophat
8  from pyms.BillerBiemann import BillerBiemann, rel_threshold, num_ions_threshold
9
10 from pyms.Peak.Function import peak_sum_area
11
12 # read the raw data as a GCMS_data object
13 andi_file = "data/gc01_0812_066.cdf"
14 data = ANDI_reader(andi_file)
15
16 im = build_intensity_matrix_i(data)
17
18 n_scan, n_mz = im.size
19
20 print("Intensity matrix size (scans, masses):", (n_scan, n_mz))
21
22 # noise filter and baseline correct
23 for ii in range(n_mz):
24     ic = im.get_ic_at_index(ii)
25     ic_smooth = savitzky_golay(ic)
26     ic_bc = tophat(ic_smooth, struct="1.5m")
27     im.set_ic_at_index(ii, ic_bc)
28
29 # Use Biller and Biemann technique to find apexing ions at a scan.
30 peak_list = BillerBiemann(im, points=9, scans=2)
31
32 # percentage ratio of ion intensity to max ion intensity
33 r = 2
34 # minimum number of ions, n
35 n = 3
36 # greater than or equal to threshold, t
37 t = 10000
38
39 # trim by relative intensity
40 pl = rel_threshold(peak_list, r)
41
42 # trim by threshold
43 new_peak_list = num_ions_threshold(pl, n, t)
44
45 print("Number of filtered peaks: ", len(new_peak_list))
46
47 # find and set areas
48 print("Peak areas")
49 print("UID, RT, height, area")
50 for peak in new_peak_list:
51     rt = peak.rt
52     # Only test interesting sub-set from 29.5 to 32.5 minutes
53     if rt >= 29.5*60.0 and rt <= 32.5*60.0:
54         # determine and set area

```

(continues on next page)

(continued from previous page)

```

55     area = peak_sum_area(im, peak)
56     peak.area = area
57
58     # print some details
59     UID = peak.UID
60     # height as sum of the intensities of the apexing ions
61     height = sum(peak.mass_spectrum.mass_spec)
62     print(UID + " ", {rt / 60.0:.2f}, {height:.2f}, {peak.area:.2f})

```

6.4.6 56

Download Source

```

1  """proc.py
2  """
3
4  from pyms.GCMS.IO.ANDI import ANDI_reader
5  from pyms.IntensityMatrix import build_intensity_matrix_i
6  from pyms.Noise.SavitzkyGolay import savitzky_golay
7  from pyms.TopHat import tophat
8
9  from pyms.BillerBiemann import BillerBiemann, rel_threshold, num_ions_threshold
10
11 from pyms.Peak.Function import peak_top_ion_areas
12
13 # read the raw data as a GCMS_data object
14 andi_file = "data/gc01_0812_066.cdf"
15 data = ANDI_reader(andi_file)
16
17 im = build_intensity_matrix_i(data)
18
19 n_scan, n_mz = im.size
20
21 print("Intensity matrix size (scans, masses):", (n_scan, n_mz))
22
23 # noise filter and baseline correct
24 for ii in range(n_mz):
25     ic = im.get_ic_at_index(ii)
26     ic_smooth = savitzky_golay(ic)
27     ic_bc = tophat(ic_smooth, struct="1.5m")
28     im.set_ic_at_index(ii, ic_bc)
29
30 # Use Biller and Biemann technique to find apexing ions at a scan.
31 peak_list = BillerBiemann(im, points=9, scans=2)
32
33 # percentage ratio of ion intensity to max ion intensity
34 r = 2
35 # minimum number of ions, n
36 n = 3
37 # greater than or equal to threshold, t
38 t = 10000
39
40 # trim by relative intensity
41 pl = rel_threshold(peak_list, r)
42

```

(continues on next page)

(continued from previous page)

```

43 # trim by threshold
44 new_peak_list = num_ions_threshold(pl, n, t)
45
46 print("Number of filtered peaks: ", len(new_peak_list))
47
48 # find and set areas
49 print("Top 5 most abundant ions for each peak ")
50
51 for peak in new_peak_list:
52     rt = peak.rt
53     # Only test interesting sub-set from 29.5 to 32.5 minutes
54     if rt >= 29.5*60.0 and rt <= 32.5*60.0:
55         # determine and set ion areas, use default num of ions =5
56         areas_dict = peak_top_ion_areas(im, peak)
57         peak.ion_areas = areas_dict
58
59         area_dict = peak.ion_areas
60         # print the top 5 ions for each peak
61         print(area_dict.keys())

```

6.4.7 64

Peak alignment with the “common ion” filtering.

[Download Source](#)

```

1  """proc.py
2  """
3
4  import os
5
6  from pyms.Experiment import load_expr
7  from pyms.DPA.PairwiseAlignment import PairwiseAlignment, align_with_tree
8  from pyms.DPA.Alignment import exprl2alignment
9  #from pyms.Peak.List.IO import store_peaks
10
11 # define the input experiments list
12 exprA_codes = [ "a0806_077", "a0806_078", "a0806_079" ]
13 exprB_codes = [ "a0806_140", "a0806_141", "a0806_142" ]
14
15 # within replicates alignment parameters
16 Dw = 2.5 # rt modulation [s]
17 Gw = 0.30 # gap penalty
18
19 # do the alignment
20 print('Aligning expt A')
21 expr_list = []
22 expr_dir = "../old demos/61a/output/"
23 for expr_code in exprA_codes:
24     file_name = os.path.join(expr_dir, expr_code + ".expr")
25     expr = load_expr(file_name)
26     expr_list.append(expr)
27 F1 = exprl2alignment(expr_list)
28 T1 = PairwiseAlignment(F1, Dw, Gw)
29 A1 = align_with_tree(T1, min_peaks=2)

```

(continues on next page)

(continued from previous page)

```

30
31 top_ion_list = A1.common_ion()
32 A1.write_common_ion_csv('output/area2.csv', top_ion_list)
33
34 print('Aligning expt B')
35 expr_list = []
36 expr_dir = "../old demos/61b/output/"
37 for expr_code in exprB_codes:
38     file_name = os.path.join(expr_dir, expr_code + ".expr")
39     expr = load_expr(file_name)
40     expr_list.append(expr)
41 F2 = exprl2alignment(expr_list)
42 T2 = PairwiseAlignment(F2, Dw, Gw)
43 A2 = align_with_tree(T2, min_peaks=2)
44
45 # between replicates alignment parameters
46 Db = 10.0 # rt modulation
47 Gb = 0.30 # gap penalty
48
49 top_ion_list = A2.common_ion()
50 A2.write_common_ion_csv('output/area1.csv', top_ion_list)
51
52 print('Aligning input {1,2}')
53 T9 = PairwiseAlignment([A1,A2], Db, Gb)
54 A9 = align_with_tree(T9)
55
56 top_ion_list = A9.common_ion()
57 A9.write_common_ion_csv('output/area.csv', top_ion_list)

```

6.4.8 A1

Download Source

```

1  """
2  proc.py
3
4  Plot detected peaks using matplotlib
5  """
6
7  import sys
8  sys.path.append("../..")
9
10 import pathlib
11
12 import matplotlib
13 matplotlib.use("TkAgg")
14 import matplotlib.pyplot as plt
15
16 from pyms.BillerBiemann import BillerBiemann, rel_threshold, num_ions_threshold
17 from pyms.Display import plot_ic, plot_peaks
18 from pyms.GCMS.IO.ANDI import ANDI_reader
19 from pyms.IntensityMatrix import build_intensity_matrix_i
20 from pyms.Noise.SavitzkyGolay import savitzky_golay
21 from pyms.TopHat import tophat
22

```

(continues on next page)

(continued from previous page)

```

23
24 data_directory = pathlib.Path(".").resolve().parent.parent / "pym-s-data"
25 # Change this if the data files are stored in a different location
26
27 output_directory = pathlib.Path(".").resolve() / "output"
28
29 # Read raw data
30 andi_file = data_directory / "MM-10.0_1_no_processing.cdf"
31 data = ANDI_reader(andi_file)
32
33 # Build Intensity Matrix
34 im = build_intensity_matrix_i(data)
35
36 # Perform pre-filtering and peak detection.
37
38 n_scan, n_mz = im.size
39
40 for ii in range(n_mz):
41     ic = im.get_ic_at_index(ii)
42     ic_smooth = savitzky_golay(ic)
43     ic_bc = tophat(ic_smooth, struct="1.5m")
44     im.set_ic_at_index(ii, ic_bc)
45
46 # Detect Peaks
47 peak_list = BillerBiemann(im, points=9, scans=2)
48
49 print("Number of peaks found: ", len(peak_list))
50
51 # Filter the peak list, first by removing all intensities in a peak less than a
52 # given relative threshold, then by removing all peaks that have less than a
53 # given number of ions above a given value
54
55 # Parameters
56 # percentage ratio of ion intensity to max ion intensity
57 percent = 2
58 # minimum number of ions, n
59 n = 3
60 # greater than or equal to threshold, t
61 cutoff = 10000
62
63 # trim by relative intensity
64 pl = rel_threshold(peak_list, percent)
65
66 # trim by threshold
67 new_peak_list = num_ions_threshold(pl, n, cutoff)
68
69 print("Number of filtered peaks: ", len(new_peak_list))
70
71 # TIC from raw data
72 tic = data.tic
73
74 # Get Ion Chromatograms for all m/z channels
75 n_mz = len(im.mass_list)
76
77 # Create a subplot
78 fig, ax = plt.subplots(1, 1)
79

```

(continues on next page)

(continued from previous page)

```

80 # Plot the peaks
81 plot_peaks(ax, new_peak_list, style="lines")
82 # Note: No idea why, but the dots for the peaks consistently appear 2e7 below the
83 # ↪ apex of the peak.
84 # As an alternative, the positions of the peaks can be shown with thin grey lines as
85 # ↪ in this example.
86 # The peak positions seem to appear OK in the other examples.
87 # See pyms-demo/scripts/Displaying_Detected_Peaks.py for a better example
88
89 # Plot the TIC
90 plot_ic(ax, tic, label="TIC")
91
92 # Plot the ICs
93 # for m in range(n_mz):
94 #     plot_ic(ax, im.get_ic_at_index(m))
95
96 # Set the title
97 ax.set_title('TIC and PyMS Detected Peaks')
98
99 # Add the legend
100 plt.legend()
101
102 # Show the plot
103 plt.show()

```

6.4.9 A2

Download Source

```

1  """proc.py
2
3  This example demonstrates processing of a GC-TOF (Leco Pegasus-ChromaTOF)
4  generated dataset.
5
6  GC-TOF data is made up of nearly 10 scans per second. As a result of this,
7  the peak detection window in the Biller Biemann algorithm has a higher value
8  as compared to the value used to process GC-Quad data.
9
10 Due to the same reason, the value of the number of scans in the Biller
11 Biemann algorithm has a higher value as compared to the value used to
12 process GC-Quad data.
13 """
14
15 import pathlib
16
17 from pyms.GCMS.IO.ANDI import ANDI_reader
18 from pyms.IntensityMatrix import build_intensity_matrix_i
19 from pyms.Noise.SavitzkyGolay import savitzky_golay
20 from pyms.TopHat import tophat
21 from pyms.Peak.Function import peak_sum_area
22 from pyms.Display import Display
23 from pyms.BillerBiemann import BillerBiemann, rel_threshold, num_ions_threshold
24
25 data_directory = pathlib.Path(".").resolve().parent.parent / "pyms-data"
26 # Change this if the data files are stored in a different location

```

(continues on next page)

(continued from previous page)

```

27
28 output_directory = pathlib.Path(".").resolve() / "output"
29
30 # from numpy import *
31
32 # read raw data
33 andi_file = data_directory / "MM-10.0_1_no_processing.cdf"
34 data = ANDI_reader(andi_file)
35
36 # Build Intensity Matrix
37 im = build_intensity_matrix_i(data)
38 n_scan, n_mz = im.size
39
40 # perform necessary pre filtering
41 for ii in range(n_mz):
42     ic = im.get_ic_at_index(ii)
43     ic_smooth = savitzky_golay(ic)
44     ic_bc = tophat(ic_smooth, struct="1.5m")
45     im.set_ic_at_index(ii, ic_bc)
46
47 # Detect Peaks
48 peak_list = BillerBiemann(im, points=15, scans=3)
49 print("Number of peaks found: ", len(peak_list))
50
51 ##### Filter peaks#####
52 # Filter the peak list,
53 # first by removing all intensities in a peak less than a given relative
54 # threshold,
55 # then by removing all peaks that have less than a given number of ions above
56 # a given value
57
58 # Parameters
59 # percentage ratio of ion intensity to max ion intensity
60 r = 2
61 # minimum number of ions, n
62 n = 2
63 # greater than or equal to threshold, t
64 t = 4000
65 # trim by relative intensity
66 pl = rel_threshold(peak_list, r)
67
68 # trim by threshold
69 new_peak_list = num_ions_threshold(pl, n, t)
70
71 print("Number of filtered peaks: ", len(new_peak_list))
72 print("Peak areas")
73 print("UID, RT, height, area")
74 for peak in new_peak_list:
75     rt = peak.rt
76
77     # determine and set area
78     area = peak_sum_area(im, peak)
79     peak.area = area
80
81     # print some details
82     UID = peak.UID
83     # height as sum of the intensities of the apexing ions

```

(continues on next page)

(continued from previous page)

```

84     height = sum(peak.mass_spectrum.mass_spec.tolist())
85     print(UID + f", {rt:.2f}, {height:.2f}, {peak.area:.2f}")
86
87     # TIC from raw data
88     tic = data.tic
89     # baseline correction for TIC
90     tic_bc = tophat(tic, struct="1.5m")
91
92     # Get Ion Chromatograms for all m/z channels
93     n_mz = len(im.mass_list)
94     ic_list = []
95
96     for m in range(n_mz):
97         ic_list.append(im.get_ic_at_index(m))
98
99     # Create a new display object, this time plot the ICs
100    # and the TIC, as well as the peak list
101    display = Display()
102    display.plot_tic(tic_bc, 'TIC BC')
103    for ic in ic_list:
104        display.plot_ic(ic)
105    display.plot_peaks(new_peak_list, 'Peaks')
106    display.do_plotting('TIC, and PyMassSpec Detected Peaks')
107    display.show_chart()

```

6.4.10 x10

An example of parallel processing of data. Shows how to loop over all ICs in an intensity matrix, and perform noise smoothing on each IC (in parallel). Please see User Guide for instructions how to run this example on multiple CPUs

[Download Source](#)

```

1  """proc.py
2  """
3
4
5  from pyms.GCMS.IO.ANDI import ANDI_reader
6  from pyms.IntensityMatrix import build_intensity_matrix_i
7  from pyms.Noise.Window import window_smooth
8
9  # read the raw data as a GCMS_data object
10 andi_file = "data/gc01_0812_066.cdf"
11 data = ANDI_reader(andi_file)
12
13 # build the intensity matrix
14 im = build_intensity_matrix_i(data)
15
16 # get the size of the intensity matrix
17 n_scan, n_mz = im.size
18 print("Size of the intensity matrix is (n_scans, n_mz):", n_scan, n_mz)
19
20 # loop over all m/z values, fetch the corresponding IC, and perform
21 # noise smoothing
22 for ii in im.iter_ic_indices():
23     print(ii+1,)

```

(continues on next page)

(continued from previous page)

```
24     ic = im.get_ic_at_index(ii)
25     ic_smooth = window_smooth(ic, window=7)
```

6.5 Contributing

6.5.1 Overview

PyMassSpec uses `tox` to automate testing and packaging, and `pre-commit` to maintain code quality.

Install `pre-commit` with `pip` and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

6.5.2 Coding style

`Yapf` is used for code formatting, and `isort` is used to sort imports.

`yapf` and `isort` can be run manually via `pre-commit`:

```
$ pre-commit run yapf -a
$ pre-commit run isort -a
```

The complete autoformatting suite can be run with `pre-commit`:

```
$ pre-commit run -a
```

6.5.3 Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6, run:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

A series of reference images for `test_Display.py` are in the “tests/baseline” directory. If these files need to be regenerated, run the following command:

```
$ pytest --mpl-generate-path="tests/baseline" tests/test_Display.py
```

6.5.4 Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

6.5.5 Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```

6.5.6 Downloading source code

The PyMassSpec source code is available on GitHub, and can be accessed from the following URL: <https://github.com/PyMassSpec/PyMassSpec>

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/PyMassSpec/PyMassSpec
```

```
Cloning into 'PyMassSpec'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a ‘zip’ file by clicking:

Clone or download → Download Zip

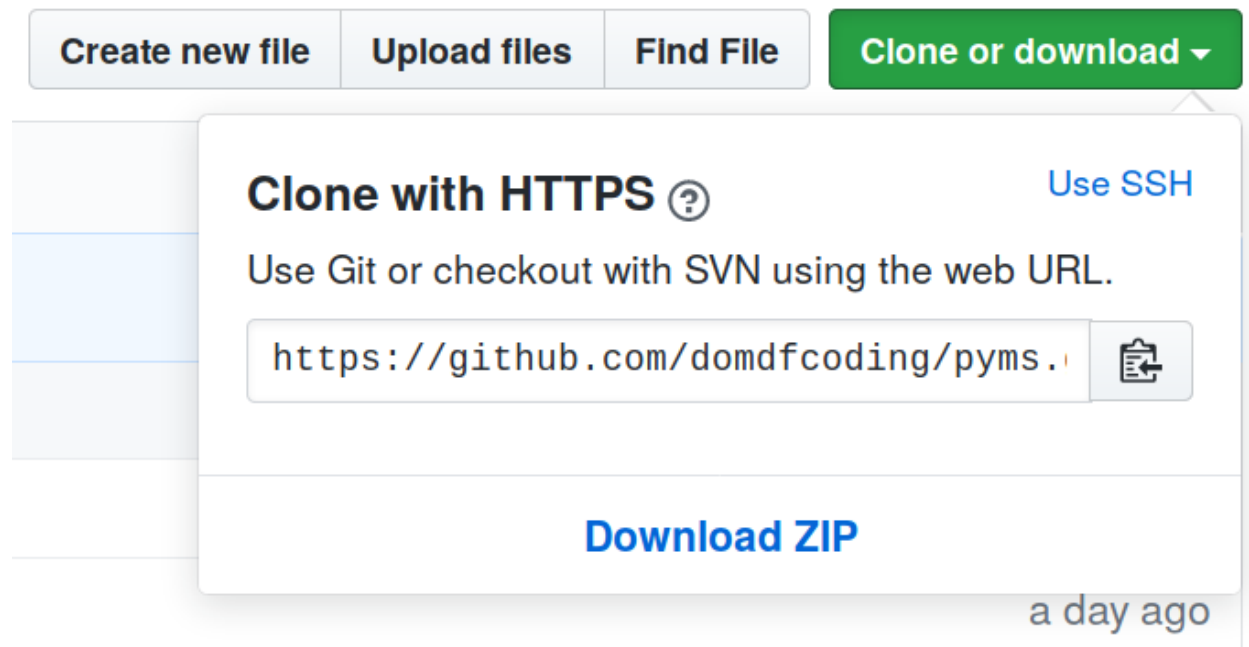


Fig. 6: Downloading a ‘zip’ file of the source code

Building from source

The recommended way to build PyMassSpec is to use `tox`:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use `pep517.build` or another **PEP 517**-compatible build tool.

6.5.7 PyMassSpec coding Style Guide

Table of Contents

- *General*
 - *Grouping commands and using newlines*
 - *File pointers*
 - *Short Comments*
- *Imports*
 - *Grouping*
 - *Import forms*
- *Naming Styles*
 - *Variable names*
 - *Module names*
 - *Class names*
 - *Exception Names*
 - *Function Names*
 - *Method Names*
 - *Internal methods and instance variables*
 - *Class-private names*
 - *Private/public class attributes*
 - *Reminder: Python names with specific meanings*
- *Docstrings*
 - *General*
 - *Packages*
 - *Modules*
 - *Functions*
 - *Classes*

This document provides specific style conventions for PyMassSpec. It should be read in conjunction with **PEP 8** “Style Guide for Python Code”, by Guido van Rossum and Barry Warsaw

General

Grouping commands and using newlines

Sort functions and class methods alphabetically, with dunder methods at the top.

Return `copy.copy` or `copy.deepcopy` only when this will not impact performance or otherwise absolutely necessary. Alternatively, use `numpy.array().tolist()`.

Organise commands into logical groups, and separate if necessary with newlines to improve readability.

Example:

```
# -- snip --
if not isinstance(file_name, str):
    raise TypeError("'file_name' must be a string")

try:
    file = CDF(file_name)
    self.__file_name = file_name
    self.__file_handle = file
except CDFError:
    error("Cannot open file '%s'" % file_name)

print(" -> Processing netCDF file '%s'" % (self.__file_name))

self.__set_min_max_mass(file)
self.__set_intensity_list(file)
# -- snip --
```

In block statements (such as for loops and if statements), do not use the blank line in a single group of statements; use one blank line to separate if the block contains more than one group of statements.

Examples:

```
# -- snip --
td_list = []
for ii in range(len(time_list) - 1):
    td = time_list[ii + 1] - time_list[ii]
    td_list.append(td)
# -- snip --
```

```
# -- snip ---
if len(time_list) > len(intensity_matrix):

    self.set_scan_index()
    scan_index_list = self.__scan_index_list

    count = 0
    while len(intensity_matrix) < len(time_list):
        count = count + 1
        scan = numpy.repeat([0], max_mass - min_mass + 1)
        intensity_matrix.insert(0, scan)
# -- snip ---
```

File pointers

Use `fp` for file pointer variables. If simultaneous use of two or more file pointers is required, use `fp1`, `fp2`, etc.

Example:

```
with open("some_file.txt", 'w', encoding="UTF-8") as fp1:
    with open("another.txt", 'w', encoding="UTF-8") as fp2:
        pass
```

Short Comments

If a comment is short, the period at the end is best omitted. Longer comments of block comments generally consist of one or more paragraphs built out of complete sentences, and each sentence should end with a period.

Imports

Grouping

Group imports as:

1. Standard library imports
2. External module imports
3. Other PyMassSpec subpackage imports
4. This subpackage imports

Separate each group by a blank line.

Import forms

For standard library modules, always import the entire module name space. i.e.

```
# stdlib
import os

...
os.path()
```

Naming Styles

Variable names

Global variable names should be prefixed with an underscore to prevent their export from the module.

For Specific variable names:

- Use `file_name` instead of `filename`
- Use `fp` for file pointer, i.e.

```
with open(file_name, 'r', encoding="UTF-8") as fp:
    pass
```

Module names

Module names should be short, starting with an uppercase letter (i.e. `Utils.py`).

Class names

Class names use the CapWords convention. Classes for internal use have a leading underscore in addition.

Exception Names

Exceptions should be handled via the function `pyms.Utils.Error.error()`.

Function Names

Function names should be lowercase, with words separated by underscores where suitable to improve readability.

Method Names

Method names should follow the same principles as the function names.

Internal methods and instance variables

Use one leading underscore only for internal methods and instance variables which are not intended to be part of the class's public interface.

Class-private names

Use two leading underscores to denote class-private names, this includes class-private methods (eg. `__privfunc()`).

Note: Python “mangles” these names with the class name: if class `Foo` has an attribute named `__a`, it cannot be accessed by `Foo.__a`. (it still could be accessed by calling `Foo._Foo__a`.)

Private/public class attributes

Public attributes should have no leading or trailing underscores. Private attributes should have two leading underscores, no trailing underscores. Non-public attributes should have a single leading underscore, no trailing underscores (the difference between private and non-public is that the former will never be useful for a derived class, while the latter might be).

Reminder: Python names with specific meanings

- `_single_leading_underscore`: weak “internal use” indicator (e.g. “from M import *” does not import objects whose name starts with an underscore).
- `single_trailing_underscore_`: used by convention to avoid conflicts with Python keyword, “Tkinter.Toplevel(master, class_='ClassName’)”.
- `__double_leading_underscore`: class-private names as of Python 1.4.
- `__double_leading_and_trailing_underscore__`: “magic” objects or attributes that live in user-controlled namespaces, e.g. `__init__`, `__import__` or `__file__`.

Docstrings

General

- All sub-packages, modules, functions, and classes must have proper Sphinx docstrings
- When designating types for `:type` and `:rtype`, use the official names from the ‘types’ package i.e. `BooleanType`, `StringType`, `FileType` etc.
- All docstrings must start with a single summary sentence concisely describing the function, and this sentence must not be terminated by a period. Additional description may follow in the form of multi-sentenced paragraphs, separated by a blank line from the summary sentence - Leave one blank line above and below the docstring
- Separate `:summary`, `:param/:type`, `:return/:rtype`, `:author` strings with one blank line

Packages

Package doctstrings are defined in `__init__.py`. This example shows top three lines of `pyms.__input__.py`:

Example:

```
"""
The root of the package pyms
"""
```

Modules

A summary for the module should be written concisely in a single sentence, enclosed above and below with lines containing only `"""`

Example:

```
"""
Provides general I/O functions
"""
```

Functions

In all functions the following Sphinx tags must be defined:

- `:param`
- `:return`
- `:author`

Other fields are optional.

The parameter and return types must be specified using type annotations per [PEP 484](#).

Example:

```
# stdlib
from typing import IO

def open_for_reading(file_name: str) -> IO:
    """
    Opens file for reading, returns file pointer

    :param file_name: Name of the file to be opened for reading

    :return: Pointer to the opened file

    :author: Jake Blues
    """
```

Classes

- The root class docstring must contain the `:author` field,

in addition to `:param` and `:return` fields for the `__init__` method. Other fields are optional. `__init__` should have no docstring.

- Methods docstrings adhere to rules for Functions. Docstrings are optional for special methods (i.e. `__len__()`, `__del__()`, etc).
- Class methods. The rules for functions apply, except that the tag `:author` does not need to be defined (if authors are given in the class docstring).

Examples:


```
class ChemStation:
    """
    ANDI-MS reader for Agilent ChemStation NetCDF files

    :param file_name: The name of the ANDI-MS file

    :author: Jake Blues
    """

    def __init__(self, file_name: str):
        pass
```


Python Module Index

p

- `pyms.Base`, 65
- `pyms.BillerBiemann`, 65
- `pyms.Display`, 68
- `pyms.DPA`, 74
 - `pyms.DPA.Alignment`, 74
 - `pyms.DPA.clustering`, 82
 - `pyms.DPA.IO`, 81
 - `pyms.DPA.PairwiseAlignment`, 78
- `pyms.Experiment`, 83
- `pyms.Gapfill`, 85
 - `pyms.Gapfill.Class`, 85
 - `pyms.Gapfill.Function`, 88
- `pyms.GCMS`, 90
 - `pyms.GCMS.Class`, 90
 - `pyms.GCMS.Function`, 94
 - `pyms.GCMS.IO`, 95
 - `pyms.GCMS.IO.ANDI`, 95
 - `pyms.GCMS.IO.JCAMP`, 95
 - `pyms.GCMS.IO.MZML`, 95
- `pyms.IntensityMatrix`, 96
- `pyms.IonChromatogram`, 108
- `pyms.json`, 119
- `pyms.Mixins`, 121
- `pyms.Noise`, 134
 - `pyms.Noise.Analysis`, 134
 - `pyms.Noise.SavitzkyGolay`, 134
 - `pyms.Noise.Window`, 135
- `pyms.Peak`, 136
 - `pyms.Peak.Class`, 136
 - `pyms.Peak.Function`, 145
 - `pyms.Peak.List`, 148
 - `pyms.Peak.List.Function`, 148
 - `pyms.Peak.List.IO`, 149
- `pyms.Simulator`, 149
- `pyms.Spectrum`, 123
- `pyms.TopHat`, 152
- `pyms.Utills`, 153
 - `pyms.Utills.IO`, 153
 - `pyms.Utills.Math`, 154
 - `pyms.Utills.Time`, 157
 - `pyms.Utills.Utills`, 158

Symbols

_C (in module *pyms.Spectrum*), 132
 _M (in module *pyms.Spectrum*), 133
 _S (in module *pyms.Spectrum*), 133
 __copy__() (*BasePeakChromatogram* method), 109
 __copy__() (*CompositeMassSpectrum* method), 125
 __copy__() (*Experiment* method), 84
 __copy__() (*ExtractedIonChromatogram* method), 113
 __copy__() (*IonChromatogram* method), 116
 __copy__() (*MassSpectrum* method), 128
 __copy__() (*Scan* method), 131
 __deepcopy__() (*Experiment* method), 84
 __eq__() (*AbstractPeak* method), 137
 __eq__() (*BaseIntensityMatrix* method), 97
 __eq__() (*BasePeakChromatogram* method), 109
 __eq__() (*CompositeMassSpectrum* method), 125
 __eq__() (*Experiment* method), 84
 __eq__() (*ExtractedIonChromatogram* method), 113
 __eq__() (*GCMS_data* method), 91
 __eq__() (*ICPeak* method), 139
 __eq__() (*IntensityMatrix* method), 102
 __eq__() (*IonChromatogram* method), 117
 __eq__() (*MassSpectrum* method), 128
 __eq__() (*Peak* method), 142
 __eq__() (*Scan* method), 131
 __len__() (*Alignment* method), 75
 __len__() (*BaseIntensityMatrix* method), 97
 __len__() (*BasePeakChromatogram* method), 109
 __len__() (*CompositeMassSpectrum* method), 125
 __len__() (*Experiment* method), 84
 __len__() (*ExtractedIonChromatogram* method), 113
 __len__() (*GCMS_data* method), 91
 __len__() (*IntensityMatrix* method), 102
 __len__() (*IonChromatogram* method), 117
 __len__() (*MassSpectrum* method), 128
 __len__() (*Scan* method), 131
 __repr__() (*GCMS_data* method), 91
 __str__() (*GCMS_data* method), 91
 __sub__() (*BasePeakChromatogram* method), 109
 __sub__() (*ExtractedIonChromatogram* method), 113
 __sub__() (*IonChromatogram* method), 117

A

AbstractPeak (class in *pyms.Peak.Class*), 136
 add_gaussc_noise() (in module *pyms.Simulator*), 150
 add_gaussc_noise_ic() (in module *pyms.Simulator*), 150
 add_gaussv_noise() (in module *pyms.Simulator*), 150
 add_gaussv_noise_ic() (in module *pyms.Simulator*), 150
 add_missing_peak() (*Sample* method), 87
 align() (in module *pyms.DPA.PairwiseAlignment*), 78
 align_with_tree() (in module *pyms.DPA.PairwiseAlignment*), 79
 aligned_peaks() (*Alignment* method), 75
 Alignment (class in *pyms.DPA.Alignment*), 74
 alignment_compare() (in module *pyms.DPA.PairwiseAlignment*), 79
 alignment_similarity() (in module *pyms.DPA.PairwiseAlignment*), 79
 ANDI_reader() (in module *pyms.GCMS.IO.ANDI*), 95
 area() (*AbstractPeak* property), 137
 area() (*ICPeak* property), 140
 area() (*Peak* property), 142
 array_as_numeric() (in module *pyms.Spectrum*), 133
 ASCII_CSV (*AsciiFiletypes* attribute), 96
 ASCII_DAT (*AsciiFiletypes* attribute), 96

B

BaseIntensityMatrix (class in *pyms.IntensityMatrix*), 96
 BasePeakChromatogram (class in *pyms.IonChromatogram*), 108
 BillerBiemann() (in module *pyms.BillerBiemann*), 65
 bounds() (*AbstractPeak* property), 137
 bounds() (*ICPeak* property), 140
 bounds() (*Peak* property), 142
 bpc() (*IntensityMatrix* property), 102
 build_intensity_matrix() (in module *pyms.IntensityMatrix*), 107

`build_intensity_matrix_i()` (in module *pyms.IntensityMatrix*), 107

C

`chromatogram()` (in module *pyms.Simulator*), 151
`ClickEventHandler` (class in *pyms.Display*), 68
`common_ion()` (Alignment method), 75
`common_ion()` (MissingPeak property), 86
`common_ion_area` (MissingPeak attribute), 86
`composite_peak()` (in module *pyms.Peak.List.Function*), 148
`CompositeMassSpectrum` (class in *pyms.Spectrum*), 124
`crop()` (CompositeMassSpectrum method), 125
`crop()` (MassSpectrum method), 128
`crop_mass()` (BaseIntensityMatrix method), 98
`crop_mass()` (IntensityMatrix method), 103
`crop_mass()` (Peak method), 143

D

`default()` (PyMassSpecEncoder method), 120
`diff()` (in module *pyms.GCMS.Function*), 94
`Display` (class in *pyms.Display*), 69
`do_plotting()` (Display method), 70
`dp()` (in module *pyms.DPA.PairwiseAlignment*), 79
`dump()` (AbstractPeak method), 138
`dump()` (BaseIntensityMatrix method), 98
`dump()` (BasePeakChromatogram method), 109
`dump()` (CompositeMassSpectrum method), 125
`dump()` (Experiment method), 84
`dump()` (ExtractedIonChromatogram method), 113
`dump()` (GCMS_data method), 91
`dump()` (ICPeak method), 140
`dump()` (IntensityMatrix method), 103
`dump()` (IonChromatogram method), 117
`dump()` (MassSpectrum method), 129
`dump()` (Peak method), 143
`dump()` (pymsBaseClass method), 65
`dump()` (Scan method), 131
`dump_object()` (in module *pyms.Utils.IO*), 153

E

`encode()` (PyMassSpecEncoder method), 120
`exact_rt` (MissingPeak attribute), 86
`Experiment` (class in *pyms.Experiment*), 83
`export_ascii()` (IntensityMatrix method), 103
`export_leco_csv()` (IntensityMatrix method), 103
`expr_code` (Alignment attribute), 75
`expr_code()` (Experiment property), 84
`exprl2alignment()` (in module *pyms.DPA.Alignment*), 77
`ExtractedIonChromatogram` (class in *pyms.IonChromatogram*), 112

F

`file2dataframe()` (in module *pyms.Gapfill.Function*), 88
`file_lines()` (in module *pyms.Utils.IO*), 153
`fill_peaks()` (in module *pyms.Peak.List.Function*), 148
`filter_min_peaks()` (Alignment method), 75
`find_mass_spectrum()` (Peak method), 143
`from_dict()` (CompositeMassSpectrum class method), 125
`from_dict()` (MassSpectrum class method), 129
`from_dict()` (Scan class method), 132
`from_jcamp()` (CompositeMassSpectrum class method), 126
`from_jcamp()` (MassSpectrum class method), 129
`from_mz_int_pairs()` (CompositeMassSpectrum class method), 126
`from_mz_int_pairs()` (MassSpectrum class method), 129
`from_spectra()` (CompositeMassSpectrum class method), 126

G

`gaussian()` (in module *pyms.Simulator*), 151
`GCMS_data` (class in *pyms.GCMS.Class*), 90
`gcms_sim()` (in module *pyms.Simulator*), 151
`get_5_largest()` (Display static method), 70
`get_area_alignment()` (Alignment method), 76
`get_highest_mz_ion()` (Alignment static method), 76
`get_ic_at_index()` (BaseIntensityMatrix method), 98
`get_ic_at_index()` (IntensityMatrix method), 103
`get_ic_at_mass()` (IntensityMatrix method), 103
`get_index_at_time()` (BaseIntensityMatrix method), 98
`get_index_at_time()` (BasePeakChromatogram method), 110
`get_index_at_time()` (ExtractedIonChromatogram method), 113
`get_index_at_time()` (GCMS_data method), 92
`get_index_at_time()` (GetIndexTimeMixin method), 121
`get_index_at_time()` (IntensityMatrix method), 104
`get_index_at_time()` (IonChromatogram method), 117
`get_index_of_mass()` (BaseIntensityMatrix method), 98
`get_index_of_mass()` (IntensityMatrix method), 104
`get_int_of_ion()` (Peak method), 143
`get_intensity_at_index()` (BasePeakChromatogram method), 110

[get_intensity_at_index\(\)](#)
(ExtractedIonChromatogram method), 113
[get_intensity_at_index\(\)](#) *(IonChromatogram method)*, 117
[get_intensity_for_mass\(\)](#)
(CompositeMassSpectrum method), 126
[get_intensity_for_mass\(\)](#) *(MassSpectrum method)*, 129
[get_ion_area\(\)](#) *(AbstractPeak method)*, 138
[get_ion_area\(\)](#) *(ICPeak method)*, 140
[get_ion_area\(\)](#) *(Peak method)*, 143
[get_mass_at_index\(\)](#) *(BaseIntensityMatrix method)*, 98
[get_mass_at_index\(\)](#) *(IntensityMatrix method)*, 104
[get_mass_for_intensity\(\)](#)
(CompositeMassSpectrum method), 126
[get_mass_for_intensity\(\)](#) *(MassSpectrum method)*, 129
[get_maxima_indices\(\)](#) *(in module pyme.BillerBiemann)*, 66
[get_maxima_list\(\)](#) *(in module pyme.BillerBiemann)*, 66
[get_maxima_list_reduced\(\)](#) *(in module pyme.BillerBiemann)*, 66
[get_maxima_matrix\(\)](#) *(in module pyme.BillerBiemann)*, 67
[get_mp_rt_exact_rt_dict\(\)](#) *(Sample method)*, 87
[get_ms_alignment\(\)](#) *(Alignment method)*, 76
[get_ms_at_index\(\)](#) *(BaseIntensityMatrix method)*, 99
[get_ms_at_index\(\)](#) *(IntensityMatrix method)*, 104
[get_n_largest\(\)](#) *(ClickEventHandler method)*, 69
[get_peak_alignment\(\)](#) *(Alignment method)*, 76
[get_peaks_alignment\(\)](#) *(Alignment method)*, 76
[get_scan_at_index\(\)](#) *(BaseIntensityMatrix method)*, 99
[get_scan_at_index\(\)](#) *(IntensityMatrix method)*, 104
[get_third_highest_mz\(\)](#) *(Peak method)*, 143
[get_time_at_index\(\)](#) *(BaseIntensityMatrix method)*, 99
[get_time_at_index\(\)](#) *(BasePeakChromatogram method)*, 110
[get_time_at_index\(\)](#)
(ExtractedIonChromatogram method), 114
[get_time_at_index\(\)](#) *(GCMS_data method)*, 92
[get_time_at_index\(\)](#) *(GetIndexTimeMixin method)*, 121
[get_time_at_index\(\)](#) *(IntensityMatrix method)*, 105
[get_time_at_index\(\)](#) *(IonChromatogram method)*, 117

[GetIndexTimeMixin](#) *(class in pyme.Mixins)*, 121

H

[half_area\(\)](#) *(in module pyme.Peak.Function)*, 145

I

[ic_mass\(\)](#) *(ICPeak property)*, 140
[ic_window_points\(\)](#) *(in module pyme.GCMS.Function)*, 94
[ICPeak](#) *(class in pyme.Peak.Class)*, 139
[icrop\(\)](#) *(CompositeMassSpectrum method)*, 126
[icrop\(\)](#) *(MassSpectrum method)*, 130
[import_leco_csv\(\)](#) *(in module pyme.IntensityMatrix)*, 107
[info\(\)](#) *(GCMS_data method)*, 92
[intensity_array\(\)](#) *(BaseIntensityMatrix property)*, 99
[intensity_array\(\)](#) *(BasePeakChromatogram property)*, 110
[intensity_array\(\)](#) *(ExtractedIonChromatogram property)*, 114
[intensity_array\(\)](#) *(IntensityArrayMixin property)*, 122
[intensity_array\(\)](#) *(IntensityMatrix property)*, 105
[intensity_array\(\)](#) *(IonChromatogram property)*, 118
[intensity_array_list\(\)](#) *(BaseIntensityMatrix property)*, 99
[intensity_array_list\(\)](#)
(BasePeakChromatogram property), 110
[intensity_array_list\(\)](#)
(ExtractedIonChromatogram property), 114
[intensity_array_list\(\)](#) *(IntensityArrayMixin property)*, 122
[intensity_array_list\(\)](#) *(IntensityMatrix property)*, 105
[intensity_array_list\(\)](#) *(IonChromatogram property)*, 118
[intensity_list\(\)](#) *(CompositeMassSpectrum property)*, 126
[intensity_list\(\)](#) *(MassSpectrum property)*, 130
[intensity_list\(\)](#) *(Scan property)*, 132
[intensity_matrix\(\)](#) *(BaseIntensityMatrix property)*, 99
[intensity_matrix\(\)](#) *(BasePeakChromatogram property)*, 110
[intensity_matrix\(\)](#) *(ExtractedIonChromatogram property)*, 114
[intensity_matrix\(\)](#) *(IntensityArrayMixin property)*, 122
[intensity_matrix\(\)](#) *(IntensityMatrix property)*, 105
[intensity_matrix\(\)](#) *(IonChromatogram property)*, 118

IntensityArrayMixin (class in *pyms.Mixins*), 121
 IntensityMatrix (class in *pyms.IntensityMatrix*), 101
 IntStr (in module *pyms.GCMS.Class*), 94
 invert_mass_spec() (in module *pyms.Display*), 72
 ion_area() (in module *pyms.Peak.Function*), 145
 ion_areas() (AbstractPeak property), 138
 ion_areas() (ICPeak property), 140
 ion_areas() (Peak property), 144
 IonChromatogram (class in *pyms.IonChromatogram*), 115
 is_bpc() (BasePeakChromatogram static method), 110
 is_bpc() (ExtractedIonChromatogram static method), 114
 is_bpc() (IonChromatogram static method), 118
 is_eic() (BasePeakChromatogram static method), 111
 is_eic() (ExtractedIonChromatogram static method), 114
 is_eic() (IonChromatogram static method), 118
 is_float() (in module *pyms.Utls.Math*), 155
 is_number() (in module *pyms.Utls.Utls*), 158
 is_path() (in module *pyms.Utls.Utls*), 158
 is_peak_list() (in module *pyms.Peak.List.Function*), 148
 is_sequence() (in module *pyms.Utls.Utls*), 158
 is_sequence_of() (in module *pyms.Utls.Utls*), 158
 is_str_num() (in module *pyms.Utls.Time*), 157
 is_tic() (BasePeakChromatogram method), 111
 is_tic() (ExtractedIonChromatogram method), 114
 is_tic() (IonChromatogram method), 118
 iter_ic_indices() (BaseIntensityMatrix method), 99
 iter_ic_indices() (IntensityMatrix method), 105
 iter_ms_indices() (BaseIntensityMatrix method), 100
 iter_ms_indices() (IntensityMatrix method), 105
 iter_peaks() (CompositeMassSpectrum method), 127
 iter_peaks() (MassSpectrum method), 130
 iter_peaks() (Scan method), 132
 iterencode() (PyMassSpecEncoder method), 120

J

JCAMP_reader() (in module *pyms.GCMS.IO.JCAMP*), 95

L

load_expr() (in module *pyms.Experiment*), 84
 load_object() (in module *pyms.Utls.IO*), 153
 load_peaks() (in module *pyms.Peak.List.IO*), 149
 local_size() (IntensityMatrix property), 105

M

MAD() (in module *pyms.Utls.Math*), 154
 mad_based_outlier() (in module *pyms.Utls.Math*), 155
 make_UID() (AbstractPeak method), 138
 make_UID() (ICPeak method), 140
 make_UID() (Peak method), 144
 mass() (BasePeakChromatogram property), 111
 mass() (ExtractedIonChromatogram property), 115
 mass() (IonChromatogram property), 118
 mass_list() (BaseIntensityMatrix property), 100
 mass_list() (CompositeMassSpectrum property), 127
 mass_list() (IntensityMatrix property), 105
 mass_list() (MassListMixin property), 122
 mass_list() (MassSpectrum property), 130
 mass_list() (Scan property), 132
 mass_spec() (CompositeMassSpectrum property), 127
 mass_spec() (MassSpectrum property), 130
 mass_spec() (Scan property), 132
 mass_spectrum() (Peak property), 144
 masses() (ExtractedIonChromatogram property), 115
 MassListMixin (class in *pyms.Mixins*), 122
 MassSpectrum (class in *pyms.Spectrum*), 127
 matrix_list() (BaseIntensityMatrix property), 100
 matrix_list() (BasePeakChromatogram property), 111
 matrix_list() (ExtractedIonChromatogram property), 115
 matrix_list() (IntensityArrayMixin property), 122
 matrix_list() (IntensityMatrix property), 106
 matrix_list() (IonChromatogram property), 119
 max_mass() (BaseIntensityMatrix property), 100
 max_mass() (CompositeMassSpectrum property), 127
 max_mass() (GCMS_data property), 92
 max_mass() (IntensityMatrix property), 106
 max_mass() (MassSpectrum property), 130
 max_mass() (MaxMinMassMixin property), 123
 max_mass() (Scan property), 132
 max_rt() (GCMS_data property), 92
 MaxMinMassMixin (class in *pyms.Mixins*), 122
 mean() (in module *pyms.Utls.Math*), 155
 median() (in module *pyms.Utls.Math*), 155
 median_bounds() (in module *pyms.Peak.Function*), 146
 median_outliers() (in module *pyms.Utls.Math*), 156
 merge_alignments() (in module *pyms.DPA.PairwiseAlignment*), 80
 min_mass() (BaseIntensityMatrix property), 100
 min_mass() (CompositeMassSpectrum property), 127
 min_mass() (GCMS_data property), 92
 min_mass() (IntensityMatrix property), 106

`min_mass()` (*MassSpectrum* property), 130
`min_mass()` (*MaxMinMassMixin* property), 123
`min_mass()` (*Scan* property), 132
`min_rt()` (*GCMS_data* property), 92
`missing_peak_finder()` (in module *pyms.Gapfill.Function*), 88
`missing_peaks()` (*Sample* property), 87
`MissingPeak` (class in *pyms.Gapfill.Class*), 85
module
 pyms.Base, 65
 pyms.BillerBiemann, 65
 pyms.Display, 68
 pyms.DPA, 74
 pyms.DPA.Alignment, 74
 pyms.DPA.clustering, 82
 pyms.DPA.IO, 81
 pyms.DPA.PairwiseAlignment, 78
 pyms.Experiment, 83
 pyms.Gapfill, 85
 pyms.Gapfill.Class, 85
 pyms.Gapfill.Function, 88
 pyms.GCMS, 90
 pyms.GCMS.Class, 90
 pyms.GCMS.Function, 94
 pyms.GCMS.IO, 95
 pyms.GCMS.IO.ANDI, 95
 pyms.GCMS.IO.JCAMP, 95
 pyms.GCMS.IO.MZML, 95
 pyms.IntensityMatrix, 96
 pyms.IonChromatogram, 108
 pyms.json, 119
 pyms.Mixins, 121
 pyms.Noise, 134
 pyms.Noise.Analysis, 134
 pyms.Noise.SavitzkyGolay, 134
 pyms.Noise.Window, 135
 pyms.Peak, 136
 pyms.Peak.Class, 136
 pyms.Peak.Function, 145
 pyms.Peak.List, 148
 pyms.Peak.List.Function, 148
 pyms.Peak.List.IO, 149
 pyms.Simulator, 149
 pyms.Spectrum, 123
 pyms.TopHat, 152
 pyms.Utills, 153
 pyms.Utills.IO, 153
 pyms.Utills.Math, 154
 pyms.Utills.Time, 157
 pyms.Utills.Utills, 158
`mp_finder()` (in module *pyms.Gapfill.Function*), 89
MZML (*MissingPeakFiletype* attribute), 88
`mzML_reader()` (in module *pyms.GCMS.IO.MZML*), 95

N

`n_largest_peaks()` (*CompositeMassSpectrum* method), 127
`n_largest_peaks()` (*MassSpectrum* method), 130
`name()` (*Sample* property), 87
NETCDF (*MissingPeakFiletype* attribute), 88
`normalize_mass_spec()` (in module *pyms.Spectrum*), 133
`null_mass()` (*BaseIntensityMatrix* method), 100
`null_mass()` (*IntensityMatrix* method), 106
`null_mass()` (*Peak* method), 144
`num_ions_threshold()` (in module *pyms.BillerBiemann*), 67

O

`onclick()` (*ClickEventHandler* method), 69
`onclick()` (*Display* method), 70

P

PairwiseAlignment (class in *pyms.DPA.PairwiseAlignment*), 78
Peak (class in *pyms.Peak.Class*), 141
`peak_list()` (*Experiment* property), 84
`peak_pt_bounds()` (in module *pyms.Peak.Function*), 146
`peak_sum_area()` (in module *pyms.Peak.Function*), 146
`peak_top_ion_areas()` (in module *pyms.Peak.Function*), 147
`peakalgt` (*Alignment* attribute), 76
`peakpos` (*Alignment* attribute), 77
`percentile_based_outlier()` (in module *pyms.Utills.Math*), 156
`plot_head2tail()` (in module *pyms.Display*), 72
`plot_ic()` (*Display* method), 70
`plot_ic()` (in module *pyms.Display*), 72
`plot_mass_spec()` (*Display* method), 70
`plot_mass_spec()` (in module *pyms.Display*), 73
`plot_peaks()` (*Display* method), 71
`plot_peaks()` (in module *pyms.Display*), 73
`plot_tic()` (*Display* method), 71
`position_similarity()` (in module *pyms.DPA.PairwiseAlignment*), 80
`prepare_filepath()` (in module *pyms.Utills.IO*), 154
PyMassSpecEncoder (class in *pyms.json*), 119
pyms.Base
 module, 65
pyms.BillerBiemann
 module, 65
pyms.Display
 module, 68
pyms.DPA

- module, 74
- `pyms.DPA.Alignment`
 - module, 74
- `pyms.DPA.clustering`
 - module, 82
- `pyms.DPA.IO`
 - module, 81
- `pyms.DPA.PairwiseAlignment`
 - module, 78
- `pyms.Experiment`
 - module, 83
- `pyms.Gapfill`
 - module, 85
- `pyms.Gapfill.Class`
 - module, 85
- `pyms.Gapfill.Function`
 - module, 88
- `pyms.GCMS`
 - module, 90
- `pyms.GCMS.Class`
 - module, 90
- `pyms.GCMS.Function`
 - module, 94
- `pyms.GCMS.IO`
 - module, 95
- `pyms.GCMS.IO.ANDI`
 - module, 95
- `pyms.GCMS.IO.JCAMP`
 - module, 95
- `pyms.GCMS.IO.MZML`
 - module, 95
- `pyms.IntensityMatrix`
 - module, 96
- `pyms.IonChromatogram`
 - module, 108
- `pyms.json`
 - module, 119
- `pyms.Mixins`
 - module, 121
- `pyms.Noise`
 - module, 134
- `pyms.Noise.Analysis`
 - module, 134
- `pyms.Noise.SavitzkyGolay`
 - module, 134
- `pyms.Noise.Window`
 - module, 135
- `pyms.Peak`
 - module, 136
- `pyms.Peak.Class`
 - module, 136
- `pyms.Peak.Function`
 - module, 145
- `pyms.Peak.List`

- module, 148
- `pyms.Peak.List.Function`
 - module, 148
- `pyms.Peak.List.IO`
 - module, 149
- `pyms.Simulator`
 - module, 149
- `pyms.Spectrum`
 - module, 123
- `pyms.TopHat`
 - module, 152
- `pyms.Utills`
 - module, 153
- `pyms.Utills.IO`
 - module, 153
- `pyms.Utills.Math`
 - module, 154
- `pyms.Utills.Time`
 - module, 157
- `pyms.Utills.Utills`
 - module, 158
- `pyms.Utills.Utills.signedinteger` (*in module `pyms.Utills.Utills`*), 158
- `pymsBaseClass` (*class in `pyms.Base`*), 65
- Python Enhancement Proposals
 - PEP 484, 159, 180
 - PEP 517, 175
 - PEP 8, 175

Q

- `qual_ion1()` (*MissingPeak property*), 86
- `qual_ion2()` (*MissingPeak property*), 86

R

- `read_expr_list()` (*in module `pyms.Experiment`*), 85
- `reduce_mass_spectra()` (*BaseIntensityMatrix method*), 100
- `reduce_mass_spectra()` (*IntensityMatrix method*), 106
- `rel_threshold()` (*in module `pyms.BillerBiemann`*), 67
- `rmsd()` (*in module `pyms.Utills.Math`*), 156
- `rt()` (*AbstractPeak property*), 138
- `rt()` (*ICPeak property*), 141
- `rt()` (*MissingPeak property*), 86
- `rt()` (*Peak property*), 144
- `rt_areas()` (*Sample property*), 87

S

- `Sample` (*class in `pyms.Gapfill.Class`*), 86
- `save_chart()` (*Display method*), 71
- `save_data()` (*in module `pyms.Utills.IO`*), 154

savitzky_golay() (in module *pyms.Noise.SavitzkyGolay*), 134
 savitzky_golay_im() (in module *pyms.Noise.SavitzkyGolay*), 135
 Scan (class in *pyms.Spectrum*), 131
 scan_list() (GCMS_data property), 92
 score_matrix() (in module *pyms.DPA.PairwiseAlignment*), 80
 score_matrix_mpi() (in module *pyms.DPA.PairwiseAlignment*), 80
 sele_peaks_by_rt() (in module *pyms.Peak.List.Function*), 148
 sele_rt_range() (Experiment method), 84
 set_bounds() (AbstractPeak method), 138
 set_bounds() (ICPeak method), 141
 set_bounds() (Peak method), 144
 set_ic_at_index() (BaseIntensityMatrix method), 100
 set_ic_at_index() (IntensityMatrix method), 106
 set_ion_area() (AbstractPeak method), 138
 set_ion_area() (ICPeak method), 141
 set_ion_area() (Peak method), 144
 show_chart() (Display method), 71
 similarity (Alignment attribute), 77
 size (CompositeMassSpectrum attribute), 127
 size() (BaseIntensityMatrix property), 101
 size() (IntensityMatrix property), 106
 std() (in module *pyms.Utils.Math*), 156
 store_peaks() (in module *pyms.Peak.List.IO*), 149
 sum_maxima() (in module *pyms.BillerBiemann*), 68

T

tic() (GCMS_data property), 93
 tic() (IntensityMatrix property), 107
 time_list() (BaseIntensityMatrix property), 101
 time_list() (BasePeakChromatogram property), 111
 time_list() (ExtractedIonChromatogram property), 115
 time_list() (GCMS_data property), 93
 time_list() (IntensityMatrix property), 107
 time_list() (IonChromatogram property), 119
 time_list() (TimeListMixin property), 123
 time_step() (BasePeakChromatogram property), 111
 time_step() (ExtractedIonChromatogram property), 115
 time_step() (GCMS_data property), 93
 time_step() (IonChromatogram property), 119
 time_step_std() (GCMS_data property), 93
 time_str_secs() (in module *pyms.Utils.Time*), 157
 TimeListMixin (class in *pyms.Mixins*), 123
 top_ions() (Peak method), 144
 top_ions_v1() (in module *pyms.Peak.Function*), 147

top_ions_v2() (in module *pyms.Peak.Function*), 147
 tophat() (in module *pyms.TopHat*), 152
 tophat_im() (in module *pyms.TopHat*), 152
 treecluster() (in module *pyms.DPA.clustering*), 82
 trim() (GCMS_data method), 93

U

UID() (AbstractPeak property), 137
 UID() (ICPeak property), 139
 UID() (Peak property), 142

V

vector_by_step() (in module *pyms.Utils.Math*), 156

W

window_analyzer() (in module *pyms.Noise.Analysis*), 134
 window_sele_points() (in module *pyms.Utils.Time*), 157
 window_smooth() (in module *pyms.Noise.Window*), 135
 window_smooth_im() (in module *pyms.Noise.Window*), 135
 write() (BasePeakChromatogram method), 111
 write() (ExtractedIonChromatogram method), 115
 write() (GCMS_data method), 93
 write() (IonChromatogram method), 119
 write_common_ion_csv() (Alignment method), 77
 write_csv() (Alignment method), 77
 write_excel() (in module *pyms.DPA.IO*), 81
 write_filled_csv() (in module *pyms.Gapfill.Function*), 89
 write_filled_rt_csv() (in module *pyms.Gapfill.Function*), 89
 write_intensities_stream() (GCMS_data method), 94
 write_ion_areas_csv() (Alignment method), 77
 write_mass_hunter_csv() (in module *pyms.DPA.IO*), 81
 write_transposed_output() (in module *pyms.DPA.IO*), 81