DIMSPy

Release 2.0.0

Ralf Weber, Jiarui (Albert) Zhou

Apr 27, 2020

CONTENTS

Python package for processing direct-infusion mass spectrometry-based metabolomics and lipidomics data

CHAPTER

ONE

CONTENTS

1.1 Installation

1.1.1 Conda (recommended)

Install Miniconda, follow the steps described here

Start the `conda prompt`

- Windows: Open the `Anaconda Prompt` via the Start menu
- macOS or Linux: Open a `Terminal`

Create a dimspy specific `conda` environment. This will install a the dependencies required to run `dimspy`:

```
$ conda create --yes --name dimspy -c conda-forge -c bioconda -c computational-
↪metabolomics
```

Note:

- The installation process will take a few minutes.
- Feel free to use a different name for the Conda environment

You can use the following command to remove a conda environment:

```
$ conda env remove -y --name dimspy
```

This is only required if something has gone wrong in the previous step.

Activate the `dimspy` environment:

```
$ conda activate dimspy
```

To test your `dimspy` installation, in your Conda Prompt, run the command:

```
$ dimspy --help
```

or:

```
$ python
import dimspy
```

Close and deactivate the `dimspy` environment when you're done:

```
$ conda deactivate
```

1.1.2 PyPi

Install the current release of `dimspy` with `pip`:

```
$ pip install dimspy
```

Note:

- The installation process will take a few minutes.

To upgrade to a newer release use the `--upgrade` flag:

```
$ pip install --upgrade dimspy
```

If you do not have permission to install software systemwide, you can install into your user directory using the `--user` flag:

```
$ pip install --user dimspy
```

Alternatively, you can manually download `dimspy` from GitHub or PyPI. To install one of these versions, unpack it and run the following from the top-level source directory using the Terminal:

```
$ pip install .
```

1.1.3 Testing

DIMSpy uses the Python `pytest` testing package. You can learn more about pytest on their homepage.

1.2 API reference

1.2.1 tools

dimspy.tools.**process_scans**(*source: str*, *function_noise: str*, *snr_thres: float*, *ppm: float*, *min_fraction: Optional[float] = None*, *rsd_thres: Optional[float] = None*, *min_scans: int = 1*, *filelist: Optional[str] = None*, *skip_stitching: bool = False*, *remove_mz_range: list = None*, *ringing_thres: Optional[float] = None*, *filter_scan_events: Dict = None*, *report: Optional[str] = None*, *block_size: int = 5000*, *ncpus: int = None*)

Extract, filter and average spectral data from input .RAW or .mzML files and generate a single mass spectral peaklist (object) for each of the data files within a directory or defined in the 'filelist' (if provided).

Warning: When using .mzML files generated using the Proteowizard tool, SIM-type scans will only be treated as spectra if the 'simAsSpectra' filter was set to true during the conversion process: *msconvert.exe example.raw* **–simAsSpectra** *–64 –zlib –filter "peakPicking true 1-"*

Parameters

- **source** – Path to a set/directory of .raw or .mzML files
- **function_noise** – Function to calculate the noise from each scan. The following options are available:
  - **median** - the median of all peak intensities within a given scan is used as the noise value.
  - **mean** - the unweighted mean average of all peak intensities within a given scan is used as the noise value.
  - **mad (Mean Absolute Deviation)** - the noise value is set as the mean of the absolute differences between peak intensities and the mean peak intensity (calculated across all peak intensities within a given scan).
  - **noise_packets** - the noise value is calculated using the proprietary algorithms contained in Thermo Fisher Scientific's msFileReader library. This option should only be applied when you are processing .RAW files.
- **snr_thres** – Peaks with a signal-to-noise ratio (SNR) less-than or equal-to this value will be removed from the output peaklist.
- **ppm** – Maximum tolerated m/z deviation in parts per million.
- **min_fraction** – A numerical value from 0 to 1 that specifies the minimum proportion of scans a given mass spectral peak must be detected in, in order for it to be kept in the output peaklist. Here, scans refers to replicates of the same scan event type, i.e. if set to 0.33, then a peak would need to be detected in at least 1 of the 3 replicates of a given scan event type.
- **rsd_thres** – Relative standard deviation threshold - A numerical value equal-to or greater-than 0. If greater than 0, then peaks whose intensity values have a percent relative standard deviation (otherwise termed the percent coefficient of variation) greater-than this value are excluded from the output peaklist.
- **min_scans** – Minimum number of scans required for each *m/z* window or event within a raw/mzML data file.
- **filelist** – A tab-delimited text file containing **filename** and **classLabel** information for each experimental sample. These column headers MUST be included in the first row of the table. For a standard DIMS experiment, users are advised to also include the following additional columns:
  - **injectionOrder** - integer values ranging from 1 to i, where i is the total number of independent injections performed as part of a DIMS experiment. e.g. if a study included 20 samples, each of which was injected as four independent replicates, there would be at least 20 * 4 injections, so i = 80 and the range for injection order would be from 1 to 80 in steps of 1.
  - **replicate** - integer value from 1 to r, indicating the order in which technical replicates of each study sample were injected in to the mass spectrometer, e.g. if study samples were analysed in quadruplicate, r = 4 and integer values are accordingly 1, 2, 3, 4.
  - **batch** - integer value from 1 to b, where b corresponds to the total number of batches analysed under define analysis conditions, for any given experiment. e.g. : if 4 independent plates of polar extracts were analysed in the positive ionisation mode, then valid values for batch are 1, 2, 3 and 4.

  This filelist may include additional columns, e.g. additional metadata relating to study samples. Ensure that columns names do not conflict with existing column names.

- **skip_stitching** – Selected Ion Monitoring (SIM) scans with overlapping scan ranges can be "stitched" together in to a pseudo-spectrum. This is achieved by setting this parameter to False (default).
- **remove_mz_range** – This option allows for specific m/z regions of the output peaklist to be deleted, this option may be useful for removing sections of a spectrum known to correspond to system noise peaks.
- **ringing_thres** – Fourier transform-based mass spectra often contain peaks (ringing artefacts) around spectral features that require removal. This threshold is a positive float indicating the required relative intensity a peak must exceed (with reference to the largest peak in a cluster of peaks) in order to be retained.
- **filter_scan_events** – Include or exclude specific scan events, by default all ALL scan events will be included. To include or exclude specific scan events use the following format of a dictionary.

```
>>> {"include":[[100, 300, "sim"]]} or {"include":[[100, 1000,
↪"full"]]}
```

- **report** – A tab-delimited text file to write measures of quality (e.g. RSD, number of peaks, etc) for each scan event processed in each .RAW or .mzML files.
- **block_size** – Number peaks in each centre clustering block.
- **ncpus** – Number of CPUs for parallel clustering. Default = None, indicating using all CPUs that are available

Returns List of peaklist objects

dimspy.tools.**replicate_filter**(*source: Union[Sequence[dimspy.models.peaklist.PeakList], str], ppm: float, replicates: int, min_peaks: int, rsd_thres: Optional[float] = None, filelist: Optional[str] = None, report: Optional[str] = None, block_size: int = 5000, ncpus: int = None*)

Peaks from each technical replicate (for a given study sample) are aligned using a one-dimensional hierarchical clustering procedure (applied on the mass-to-charge level). Peaks are aligned only if the difference in their mass-to-charge ratios, when divided by the average of their mass-to-charge ratios and multiplied by 1×10^6 (i.e. when measured in units of parts-per-million, ppm), is less-than or equal-to the user-defined 'ppm error tolerance'. After alignment, a set of user-defined filters are applied to retain only those peaks that:

- occur in equal-to or more-than the user-defined 'Number of technical replicates a peak has to be present in', i.e. if set to 2, then a peak must be detected in at least two of the replicate analyses, **and/or**
- have relative standard deviation (measured in %; may otherwise be referred to as the percent coefficient of variation) of intensity values, across technical replicates, that is equal-to or less-than the user-defined 'relative standard deviation threshold' (if defined, otherwise ignored).

Warning: When the parameter "number of technical replicates for each sample" is set to a value less-than the total number of technical replicates actually acquired for each study sample, this tool will automatically determine which combination of technical replicates to combine. See the parameter description (below) for further details.

Parameters

- **source** – A list of processed peaklist objects generated by 'process_scans' or path to .hdf5 file
- **ppm** – Maximum tolerated m/z deviation in parts per million.

- **replicates** – Number of technical replicates for each sample - the total number of technical replicates acquired for each study sample. This value must be set to the lowest number of technical replicates acquired for ANY of the study samples, or alternatively, may be set to the minimum number of replicates the user would like to select from the total number of technical replicates for a biological sample.
- **min_peaks** – Minimum number of technical replicates a peak has to be present in. For a given biological sample, the number of replicates that will be used to generate the replicate-filtered peaklist. If this parameter is set to a value less-than the total number of technical replicates acquired for each biological sample, it will automatically determines which combination of technical replicates yields the best overall rank. Otherwise, all technical replicates are used. Ranking of the combinations of technical replicates is based on the average of the following three scores:
  - score 1: peak count / peak count present in n-out-n (e.g. 3-out-of-3)
  - score 2: peak count present in x-out-of-n (e.g. 3-out-of-3) / MAX peak count present in x-out-of-n across sets of replicates
  - score 3: RSD categories (0-5 (score=1.0), 5-10 (score=0.9), 10-15 (score=0.8), etc)
- **rsd_thres** – Relative standard deviation threshold - a numerical value from 0 upwards that defines the acceptable percentage relative standard deviation (otherwise termed the percent coefficient of variation) of a peak's intensity across technical replicates. Peaks are removed from the output 'replicate-filtered' peaklist if this condition is not met. Set to None to skipe this filter.
- **filelist** – A tab-delimited text file containing **filename** and **classLabel** information for each experimental sample. There is no need to provide a filelist again if this has been done already as part of one of the previous processing steps (i.e. see process scans or replicate filter) - except if specific samples need to be excluded. These column headers MUST be included in the first row of the table. For a standard DIMS experiment, users are advised to also include the following additional columns:
  - **injectionOrder** - integer values ranging from 1 to i, where i is the total number of independent injections performed as part of a DIMS experiment. e.g. if a study included 20 samples, each of which was injected as four independent replicates, there would be at least 20 * 4 injections, so i = 80 and the range for injection order would be from 1 to 80 in steps of 1.
  - **replicate** - integer value from 1 to r, indicating the order in which technical replicates of each study sample were injected in to the mass spectrometer, e.g. if study samples were analysed in quadruplicate, r = 4 and integer values are accordingly 1, 2, 3, 4.
  - **batch** - integer value from 1 to b, where b corresponds to the total number of batches analysed under define analysis conditions, for any given experiment. e.g. : if 4 independent plates of polar extracts were analysed in the positive ionisation mode, then valid values for batch are 1, 2, 3 and 4.

  This filelist may include additional columns, e.g. additional metadata relating to study samples. Ensure that columns names do not conflict with existing column names.
- **report** – A tab-delimited text file to write measures of quality (e.g. RSD, number of peaks, etc) for each processed 'replicate-filtered' peaklist.
- **block_size** – Number peaks in each centre clustering block.
- **ncpus** – Number of CPUs for parallel clustering. Default = None, indicating using all CPUs that are available

Returns List of peaklist objects

dimspy.tools.**align_samples**(*source: Union[Sequence[dimspy.models.peaklist.PeakList], str], ppm: float, filelist: Optional[str] = None, block_size: int = 5000, ncpus: int = None*)

Study samples (i.e. PeakList Objects) are aligned to create PeakMatrix object. The PeakMatrix object comprises of a table, with samples along one axis and the mass-to-charge ratios of detected mass spectral peaks along the opposite axis. At the intersection of sample and mass-to-charge ratio, the intensity is given for a specific peak in a specific sample (if no intensity recorded, then 'nan' is inserted).

Parameters

- **source** – A list of processed peaklist objects generated by 'process_scans' and/or 'replicate_filter', or path to .hdf5 file.
- **ppm** – Maximum tolerated m/z deviation in parts per million.
- **filelist** – A tab-delimited text file containing **filename** and **classLabel** information for each experimental sample. There is no need to provide a filelist again if this has been done already as part of one of the previous processing steps (i.e. see process scans or replicate filter) - except if specific samples need to be excluded. These column headers MUST be included in the first row of the table.

  This filelist may include additional columns, e.g. additional metadata relating to study samples. Ensure that column names do not conflict with existing column names.
- **block_size** – Number peaks in each centre clustering block.
- **ncpus** – Number of CPUs for parallel clustering. Default = None, indicating using all CPUs that are available

Returns PeakMatrix object

dimspy.tools.**blank_filter**(*peak_matrix: Union[dimspy.models.peak_matrix.PeakMatrix, str], blank_label: str, min_fraction: float = 1.0, min_fold_change: float = 1.0, function: str = 'mean', rm_samples: bool = True, labels: Optional[str] = None*)

Parameters

- **peak_matrix** – PeakMatrix object
- **blank_label** – Label for the blank samples - a string indicating the name of the class to be used for filtering (e.g. blank), i.e. the "reference" class. This string must have been included in the "classLabel" column of the metadata file associated with the process_sans or replicate_filter function(s).
- **min_fraction** – A numeric value ranging from 0 to 1. Setting this value to None or 0 will skip this filtering step. A value greater than 0 requires that for each peak in the peak intensity matrix, at least this proportion of non-reference samples have to have an intensity value that exceeds the product of: (A) the average intensity of "reference" class intensities and (B) the user-defined "min_fold_change". If this condition is not met, the peak is removed from the peak intensity matrix.
- **min_fold_change** – A numeric value from 0 upwards. When minimum fraction filtering is enabled, this value defines the minimum required ratio between the intensity of a peak in a "non-reference" sample and the average intensity of the "reference" sample(s). Peaks with ratios exceeding this threshold are considered to have been reliably detected in a "non-reference" sample.
- **function** – Function to calculate the 'reference' intensity
  - **mean** - corresponds to using the non-weighted average of "reference" sample peak intensities (NA values are ignored) in calculating the "reference" to "non-reference" peak

intensity ratio.

- **median** - corresponds to using the median of "reference" sample peak intensities (NA values are ignored) in calculating the "reference" to "non-reference" peak intensity ratio.
- **max** corresponds to the use of the maximum intensity among "reference" sample peak intensities (NA values are ignored) in calculating the "reference" to "non-reference" peak intensity ratio.

- **rm_samples** – Remove blank samples from the output peak matrix: * **True** - samples belonging to the user-defined "reference" class are removed from the output peak matrix * **False** - samples belonging to the user-defined "reference" class are retained in the output peak matrix.
- **labels** – Path to the metadata file

Returns PeakMatrix object

dimspy.tools.**sample_filter**(*peak_matrix: Union[dimspy.models.peak_matrix.PeakMatrix, str], min_fraction: float, within: bool = False, rsd_thres: Optional[float] = None, qc_label: Optional[str] = None, labels: Optional[str] = None*)

Removes peaks from the input PeakMatrix object (or .hdf5 file that were detected in fewer-than a user-defined minimum number of study samples.

There are many and varied reasons why a peak may not have been detected in all study samples, including:

- due to having an intensity (concentration) close to the signal-to-noise limit of the system;
- due to having been present in only one of the study classes (e.g. a drug administered to the 'treatment' class samples);
- due to ion suppression/enhancement effects in the mass spectrometer source region; etc.

Parameters

- **peak_matrix** – PeakMatrix object or path to .hdf5 file
- **min_fraction** – Minimum fraction - a numeric value between 0 and 1 indicating the proportion of study samples in which a peak must have a recorded intensity value in order for it to be retained in the output peak intensity matrix; e.g. 0.5 means that at least 50% of samples (whether assessed across all classes, or within each class individually) must have a recorded intensity value for a specific peak in order for it to be retained in the output peak matrix.
- **within** – Apply sample filter within each sample class
  - **False** - check across ALL classes simultaneously whether greater-than the user-defined "Minimum fraction" of samples contained an intensity value for a specific mass spectral peak.
  - **True** - check within EACH class separately whether greater-than the user-defined "Minimum fraction" of samples contained an intensity value for a specific mass spectral peak.

> **Warning:** if in ANY class a peak is detected in greater-than the user-defined minimum fraction of samples, then the peak is retained in the output peak matrix. For classes in which this condition is not met, the peak intensity recorded for that peak (if any) will still be presented in the output peak matrix. If no peak intensity was recorded in a sample, then a '0' is inserted in to the peak matrix.

- **rsd_thres** – Relative standard deviation threshold - A numerical value equal-to or greater-than 0. If greater than 0, then peaks whose intensity values have a percent relative standard deviation (otherwise termed the percent coefficient of variation) greater-than this value are excluded from the output PeakMatrix object.
- **qc_label** – Label for the QC samples - a string indicating the name of the class to be used for filtering, i.e. the "reference" class. This string must have been included in the "classLabel" column of the metadata file associated with the process_sans or replicate_filter function(s).
- **labels** – Path to a metadata file

Returns PeakMatrix object

dimspy.tools.**missing_values_sample_filter**(*peak_matrix: dimspy.models.peak_matrix.PeakMatrix*, *max_fraction: float*)

Removes study samples with greater-than a user-defined "Maximum percentage of missing values" from the peak intensity matrix. A missing value is defined as the absence of a recorded peak intensity value for a specific mass spectral peak, in a specific study sample.

Samples with large numbers of missing values are often observed where a failed mass spectral acquisition has occurred, the reasons for which are many and diverse.

Parameters

- **peak_matrix** – PeakMatrix object
- **max_fraction** –

  Maximum percentage of missing values (REQUIRED; default = 0.8) - a numeric value ranging from 0 to 1 (decimal representation of percentage), where:

  – A value of 0 (i.e. 0%) corresponds to a very harsh filtering procedure, in which only those samples with zero missing values are retained in the output peak matrix.

  – A value of 1 (i.e. 100%) corresponds to a very liberal filtering procedure, in which samples with as many as 100% missing values will be retained in the output peak matrix.

Returns PeakMatrix object

dimspy.tools.**remove_samples**(*obj: Union[dimspy.models.peak_matrix.PeakMatrix, Sequence[dimspy.models.peaklist.PeakList]], sample_names: list*)

Remove samples from a PeakMatrix or list of PeakLists

Parameters

- **obj** – PeakMatrix object or List of PeakList objects
- **sample_names** – List of sample names (Peaklist IDs)

Returns PeakMatrix object or List of Peaklist Objects

dimspy.tools.**hdf5_peak_matrix_to_txt**(*filename: str*, *path_out: str*, *attr_name: str = 'intensity'*, *rsd_tags: tuple = ()*, *delimiter: str = '\t'*, *samples_in_rows: bool = True*, *comprehensive: bool = False*, *compatibility_mode: bool = False*)

Converts a .hdf5 file, containing a peak intensity matrix, to an user-friendly .tsv (tab-separated values) file.

Parameters

- **filename** – Path to the .hdf5 file to read from.

- **path_out** – Path to a text file to write to.
- **attr_name** – The Peak Matrix should contain Intensity|m/z|SNR| values
- **rsd_tags** – Calculate RDS values for the following sample classes (e.g. QC, control)
- **delimiter** – Values on each line of the file are separated by this character.
- **samples_in_rows** – Should the rows or columns represent the samples?
- **comprehensive** – Comprehensive Peak Matrix (e.g. m/z and intensity, rsd, missing values).
- **compatibility_mode** – Set to True to read .hdf5 files from dimspy < v2.0 exported .hdf5 files

dimspy.tools.**hdf5_peaklists_to_txt**(*filename: str*, *path_out: str*, *delimiter: str = '\t'*, *compatibility_mode: bool = False*)

Converts a .hdf5 file, containing a list peaklists, to user-friendly .tsv (tab-separated values) files.

Parameters

- **filename** – Path to the .hdf5 file to read from.
- **path_out** – Path to directory to write to.
- **delimiter** – Values on each line of the file are separated by this character.
- **compatibility_mode** – Set to True to read .hdf5 files exported using dimspy < v2.0.

dimspy.tools.**merge_peaklists**(*source: Sequence[dimspy.models.peaklist.PeakList]*, *filelist: Optional[str] = None*)

Extracts and exports specific PeakList object from one or more list or one or more .hdf5 files, to one or more lists or .hdf5 files. If more-than one .hdf5 file is exported, users can control which subset of peaklists are exported to which list.

Parameters

- **source** – List or tuple of Peaklist objects, or .hdf5 files
- **filelist** – A tab-delimited text file containing metadata to determine which peaklists are exported together:

  Example of a filelist - the optional multilist column determines which peaklists are exported together.

filename	classLabel	replicate	batch	injectionOrder	multilist	[...]
sample_rep1.raw	sample	1	1	1	1	[...]
sample_rep2.raw	sample	2	1	2	1	[...]
sample_rep3.raw	sample	3	1	3	1	[...]
sample_rep4.raw	sample	4	1	4	1	[...]
blank_rep1.raw	blank	1	1	5	2	[...]
blank_rep2.raw	blank	2	1	6	2	[...]
blank_rep3.raw	blank	3	1	7	2	[...]
blank_rep4.raw	blank	4	1	8	2	[...]
...	...	...	...	...	...	[...]

Returns Nested lists of Peaklist objects (e.g. [[pl_01, pl_02], [pl_03, pl_04, pl05]]

dimspy.tools.**partition**(*alist: list*, *indices: list*)

Divide separated lists into nested sublists

Parameters

- **alist** – List
- **indices** – Indices

Returns Nested List

dimspy.tools.**load_peaklists**(*source: Sequence[dimspy.models.peaklist.PeakList]*)
Load a set of processed PeakLists

Parameters **source** – list of Peaklist objects, .hdf5 file, or path to a directory

Returns List of Peaklist Objects

dimspy.tools.**create_sample_list**(*source: Union[Sequence[dimspy.models.peaklist.PeakList], dimspy.models.peak_matrix.PeakMatrix], path_out: str, delimiter: str = '\t'*)
Create a sample list based on a existing list of PeakList Objects or PeaMatrix Object.

Parameters

- **source** – List of PeakList objects or PeakMatrix object
- **path_out** – Path to a text file text file to write to.
- **delimiter** – Values on each line of the file are separated by this character.

1.2.2 metadata

dimspy.metadata.**count_ms_types**(*hs: list*) → int
Count the number of unique ms types

Parameters **hs** – List of headers or filter strings

Returns Count

dimspy.metadata.**count_scan_types**(*hs: list*) → int
Count the number of unique scan types

Parameters **hs** – List of headers or filter strings

Returns Count

dimspy.metadata.**idxs_reps_from_filelist**(*replicates: list*)

Parameters **replicates** –

Returns

dimspy.metadata.**interpret_method**(*mzrs: list*)
Interpret and define type of method

Parameters **mzrs** – Nested list of m/z ranges / windows

Returns Type of MS method

dimspy.metadata.**mode_type_from_header**(*h: str*) → str
Extract scan mode from the header of filter string

Parameters **h** – header or filter string

Returns Scan type (e.g. p = profile, c = centroid)

dimspy.metadata.**ms_type_from_header**(*h: str*) → str
Extract the ms type from header or filter string

Parameters **h** – header or filter string

Returns ms type (e.g. FTMS and ITMS)

dimspy.metadata.**mz_range_from_header**(*h: str*) → Sequence[float]
Extract m/z range from header or filter string

Parameters **h** – Header or filter string

Returns m/z range

dimspy.metadata.**scan_type_from_header**(*h: str*) → str
Extract the scan type from the header of filter string

Parameters **h** – header or filter string

Returns Scan type (e.g. full or sim)

dimspy.metadata.**to_int**(*x*)

Parameters **x** – Value to convert to int

Returns Value as int (or False if conversion not possible)

dimspy.metadata.**update_labels**(*pm: dimspy.models.peak_matrix.PeakMatrix*, *fn_tsv: str*) → dimspy.models.peak_matrix.PeakMatrix
Update Sample labels PeakMatrix object :param pm: peakMatrix Object :param fn_tsv: Path to tab-separated file :return: peakMatrix Object

dimspy.metadata.**update_metadata_and_labels**(*peaklists: Sequence[dimspy.models.peaklist.PeakList]*, *fl: Dict*)
Update metadata

Parameters

- **peaklists** – List of peaklist Objects
- **fl** – Dictionary with meta data

Returns List of peaklist objects

dimspy.metadata.**validate_metadata**(*fn_tsv: str*) → collections.OrderedDict
Check and validate metadata within a tab-separated file

Parameters **fn_tsv** – Path to tab-separated file

Returns Dictionary

1.2.3 models

peaklist

class dimspy.models.peaklist.**PeakList**(*ID: str*, *mz: Sequence[float]*, *intensity: Sequence[float]*, ***metadata*)
Bases: object

The PeakList class.

Stores mass spectrometry peaks list data. It requires an ID, mz values, and intensities. It can store extra peak attributes e.g. SNRs, and peaklist tags and metadata. It utilises the automatically managed flags to "remove" or "retain" peaks without actually delete them. Therefore the filterings on the peaks are traceable.

Parameters

- **ID** – The ID of the peaklist data, unique string or integer value is recommended

- **mz** – Mz values of all the peaks. Must in the ascending order
- **intensity** – Intensities of all the peaks. Must have the same size as mz
- **kwargs** – Key-value pairs of the peaklist metadata

```
>>> mz_values = np.random.uniform(100, 1200, size = 100)
>>> int_values = np.random.normal(60, 10, size = 100)
>>> peaks = PeakList('dummy', mz_values, int_values, description = 'a dummy␣
↪peaklist')
```

Internally the peaklist data is stored by using numpy structured array namely the attribute talbe (this may change in the future):

mz	intensity	snr	snr_flag	...	*flags**
102.5	21.7	10.5	True	...	True
111.7	12.3	5.1	False		False
126.3	98.1	31.7	True		True
133.1	68.9	12.6	True		True
...					

Each column is called an attribute. The first two attributes are fixed as "mz" and "intensity". They cannot be added or removed as the others. The last "attribute" is the "flags", which is fact stored separately. The "flags" column is calculated automatically according to all the manually set flag attributes, e.g., the "snr_flag". It can only be changed by the class itself. The unflagged peaks are considered as "removed". They are kept internally mainly for visualization and tracing purposes.

Warning: Removing a flag attribute may change the "flags" column, and cause the unflagged peaks to be flagged again. As most the processes are applied only on the flagged peaks, these peaks, if the others have gone through such process, may have incorrect values.

In principle, setting a flag attribute should be considered as an irreversible process.

property ID
Property of the peaklist ID.

Getter Returns the peaklist ID

Setter Set the peaklist ID

Type Same as input ID

add_attribute(*attr_name: str*, *attr_value: Sequence*, *attr_dtype: Union[Type, str, None] = None*, *is_flag: bool = False*, *on_index: Optional[int] = None*, *flagged_only: bool = True*, *invalid_value=nan*)
Adds an new attribute to the PeakList attribute table.

Parameters

- **attr_name** – The name of the new attribute, must be a string
- **attr_value** – The values of the new attribute. It's size must equals to PeakList.size (if flagged_only == True), or PeakList.full_size (if flagged_only == False)
- **attr_dtype** – The data type of the new attribute. If it is set to None, the PeakList will try to detect the data type based on attr_value. If the detection failed it will take the "object" type. Default = None

- **is_flag** – Whether the new attribute is a flag attribute, i.e., will be used in flags calculation. Default = False
- **on_index** – Insert the new attribute on a specific column. It can't be 0 or 1, as the first two attributes are fixed as mz and intensity. Setting to None means to put it to the last column. Default = None
- **flagged_only** – Whether the attr_value is set to the flagged peaks or all peaks. Default = True
- **invalid_value** – If flagged_only is set to True, this value will be assigned to the unflagged peaks. The actual value depends on the attribute data type. For instance, on a boolean attribute invalid_value = 0 will be converted to False. Default = numpy.nan

Return type PeakList object (self)

property attributes
Property of the attribute names.

Getter Returns a tuple of the attribute names

Type tuple

calculate_flags()
Re-calculates the flags according to the flag attributes.

Return type numpy array

Note: This method will be called automatically every time a flag attribute is added, removed, or changed.

---

cleanup_unflagged_peaks(*flag_name: Optional[str] = None*)
Remove unflagged peaks.

Parameters **flag_name** – Remove peaks unflagged by this flag attribute. Setting None means to remove peaks unflagged by the overall flags. Default = None

Return type PeakList object (self)

```
>>> print(peaks)
mz, intensity, intensity_flag, snr, snr_flag, flags
10, 70, True, 10, False, False
20, 60, True, 20, True, True
30, 50, False, 30, True, False
40, 40, False, 40, True, False
>>> print(peaks.cleanup_unflagged_peaks('snr_flag'))
mz, intensity, intensity_flag, snr, snr_flag, flags
20, 60, True, 20, True, True
30, 50, False, 30, True, False
40, 40, False, 40, True, False
>>> print(peaks.cleanup_unflagged_peaks())
mz, intensity, intensity_flag, snr, snr_flag, flags
20, 60, True, 20, True, True
```

copy()
Returns a deep copy of the peaklist.

Return type PeakList object

drop_attribute(*attr_name: str*)
Drops an existing attribute.

Parameters **attr_name** – The attribute name to drop. It cannot be mz, intensity, or flags

Return type PeakList object (self)

property dtable
Property of the overall attribute table.

Getter Returns the original attribute table

Type numpy structured array

> **Warning:** This property directly accesses the internal attribute table. Be careful when manipulating the data, particularly pay attention to the potential side-effects.

property flag_attributes
Property of the flag attribute names.

Getter Returns a tuple of the flag attribute names

Type tuple

property flags
Property of the flags.

Getter Returns a deep copy of the flags array

Type numpy array

property full_shape
Property of the peaklist full attributes table shape.

Getter Returns the full attibutes table shape, including the unflagged peaks

Type tuple

property full_size
Property of the peaklist full size.

Getter Returns the full peaklist size, i.e., including the unflagged peaks

Type int

get_attribute(*attr_name: str*, *flagged_only: bool = True*)
Gets values of an existing attribute.

Parameters

- **attr_name** – The attribute to get values
- **flagged_only** – Whether to return the values of flagged peaks or all peaks. Default = True

Return type numpy array

get_peak(*peak_index: Union[int, Sequence[int]]*, *flagged_only: bool = True*)
Gets values of a peak.

Parameters

- **peak_index** – The index of the peak to get values
- **flagged_only** – Whether the values are taken from the index of flagged peaks or all peaks. Default = True

Return type numpy array

has_attribute(*attr_name: str*)
Checks whether there exists an attribute in the table.

Parameters **attr_name** – The attribute name for checking

Return type bool

insert_peak(*peak_value: Sequence*)
Insert a new peak.

Parameters **peak_value** – The values of the new peak. Must contain values for all the attributes. It's position depends on the mz value, i.e., the 1st value of the input

Return type PeakList object (self)

property metadata
Property of the peaklist metadata.

Getter Returns an access interface to the peaklist metadata object

Type PeakList_Metadata object

property peaks
Property of the attribute table.

Getter Returns a deep copy of the flagged attribute table

Type numpy structured array

remove_peak(*peak_index: Union[int, Sequence[int]], flagged_only: bool = True*)
Remove an existing peak.

Parameters

- **peak_index** – The index of the peak to remove
- **flagged_only** – Whether the index is for flagged peaks or all peaks. Default = True

Return type PeakList object (self)

set_attribute(*attr_name: str*, *attr_value: Sequence*, *flagged_only: bool = True*, *unsorted_mz: bool = False*)
Sets values to an existing attribute.

Parameters

- **attr_name** – The attribute to set values
- **attr_value** – The new attribute values, It's size must equals to PeakList.size (if flagged_only == True), or PeakList.full_size (if flagged_only == False)
- **flagged_only** – Whether the attr_value is set to the flagged peaks or all peaks. Default = True
- **unsorted_mz** – Whether the attr_value contains unsorted mz values. This parameter is valid only when attr_name == "mz". Default = False

Return type PeakList object (self)

set_peak(*peak_index: int*, *peak_value: Sequence*, *flagged_only: bool = True*)
Sets values to a peak.

Parameters

- **peak_index** – The index of the peak to set values
- **peak_value** – The new peak values. Must contain values for all the attributes (not including flags)

- **flagged_only** – Whether the peak_value is set to the index of flagged peaks or all peaks. Default = True

Return type PeakList object (self)

```
>>> print(peaks)
mz, intensity, snr, flags
10, 10, 10, True
20, 20, 20, True
30, 30, 30, False
40, 40, 40, True
>>> print(peaks.set_peak(2, [50, 50, 50], flagged_only = True))
mz, intensity, snr, flags
10, 10, 10, True
20, 20, 20, True
30, 30, 30, False
50, 50, 50, True
>>> print(peaks.set_peak(2, [40, 40, 40], flagged_only = False))
mz, intensity, snr, flags
10, 10, 10, True
20, 20, 20, True
40, 40, 40, False
50, 50, 50, True
```

property shape
Property of the peaklist attributes table shape.

Getter Returns the attibutes table shape, i.e., peaks number x attributes number. The "flags" column does not count

Type tuple

property size
Property of the peaklist size.

Getter Returns the flagged peaklist size

Type int

sort_peaks_order()
Sorts peaklist mz values into ascending order.

Note: This method will be called automatically every time the mz values are changed.

property tags
Property of the peaklist tags.

Getter Returns an access interface to the peaklist tags object

Type PeakList_Tags object

to_df()
Exports peaklist attribute table to Pandas DataFrame, including the flags.

Return type pd.DataFrame

to_dict(*dict_type: Callable[[Sequence], Mapping] = <class 'collections.OrderedDict'>*) → Mapping
Exports peaklist attribute table to a dictionary (mappable object), including the flags.

Parameters **dict_type** – Result dictionary type, Default = OrderedDict

Return type list

to_list()
Exports peaklist attribute table to a list, including the flags.

Return type list

to_str(*delimiter: str = ', '*)
Exports peaklist attribute table to a string, including the flags. It can also be used inexplicitly.

Return type str

peaklist_metadata

class dimspy.models.peaklist_metadata.**PeakList_Metadata**
Bases: dict

The PeakList_Metadata class.

Dictionary-like container for PeakList metadata storage.

Parameters

- **args** – Iterable object of key-value pairs
- **kwargs** – Metadata key-value pairs

```
>>> PeakList_Metadata([('name', 'sample_1'), ('qc', False)])
>>> PeakList_Metadata(name = 'sample_1', qc = False)
```

metadata attributes can be accessed in both dictionary-like and property-like manners.

```
>>> meta = PeakList_Metadata(name = 'sample_1', qc = False)
>>> meta['name']
sample_1
>>> meta.qc
False
>>> del meta.qc
>>> meta.has_key('qc')
False
```

Warning: The *__getattr__*, *__setattr__*, and *__delattr__* methods are overrided. **DO NOT** assign a metadata object to another metadata object, e.g., metadata.metadata.attr = value.

peaklist_tags

class dimspy.models.peaklist_tags.**PeakList_Tags**(**args*, ***kwargs*)
Bases: object

The PeakList_Tags class.

Container for both typed and untyped tags. This class is mainly used in PeakList and PeakMatrix classes for sample filtering. For a PeakList the tag types must be unique, but not the tag values (unless they are untyped). For instance, PeakList can have tags batch = 1 and plate = 1, but not batch = 1 and batch = 2, or (untyped) 1 and (untyped) 1. Single value will be treated as untyped tag.

Parameters

- **args** – List of untyped tags

- **kwargs** – List of typed tags. Only one tag value can be assigned to a specific tag type

```
>>> PeakList_Tags('untyped_tag1', Tag('untyped_tag2'), Tag('typed_tag', 'tag_type
↪'))
>>> PeakList_Tags(tag_type1 = 'tag_value1', tag_type2 = 'tag_value2')
```

add_tag(*tag: Union[int, float, str, dimspy.models.peaklist_tags.Tag], tag_type: Optional[str] = None*)
Adds typed or untyped tag.

Parameters

- **tag** – Tag or tag value to add
- **tag_type** – Type of the tag value

```
>>> tags = PeakList_Tags()
>>> tags.add_tag('untyped_tag1')
>>> tags.add_tag(Tag('typed_tag1', 'tag_type1'))
>>> tags.add_tag(tag_type2 = 'typed_tag2')
```

drop_all_tags()
Drops all tags, both typed and untyped.

drop_tag(*tag: Union[int, float, str, dimspy.models.peaklist_tags.Tag], tag_type: Optional[str] = None*)
Drops typed and untyped tag.

Parameters

- **tag** – Tag or tag value to drop
- **tag_type** – Type of the tag value

```
>>> tags = PeakList_Tags('untyped_tag1', tag_type1 = 'tag_value1')
>>> tags.drop_tag(Tag('tag_value1', 'tag_type1'))
>>> print(tags)
untyped_tag1
```

drop_tag_type(*tag_type: Optional[str] = None*)
Drops the tag with the given type.

Parameters **tag_type** – Tag type to drop, None (untyped) may drop multiple tags

has_tag(*tag: Union[int, float, str, dimspy.models.peaklist_tags.Tag], tag_type: Optional[str] = None*)
Checks whether there exists a specific tag.

Parameters

- **tag** – The tag for checking
- **tag_type** – The type of the tag

Return type bool

```
>>> tags = PeakList_Tags('untyped_tag1', Tag('tag_value1', 'tag_type1'))
>>> tags.has_tag('untyped_tag1')
True
>>> tags.has_tag('typed_tag1')
False
>>> tags.has_tag(Tag('tag_value1', 'tag_type1'))
True
>>> tags.has_tag('tag_value1', 'tag_type1')
True
```

has_tag_type(*tag_type: Optional[str] = None*)
Checks whether there exists a specific tag type.

Parameters **tag_type** – The tag type for checking, None indicates untyped tags

Return type bool

tag_of(*tag_type: Optional[str] = None*)
Returns tag value of the given tag type, or tuple of untyped tags if tag_type is None.

Parameters **tag_type** – Valid tag type, None for untyped tags

Return type *Tag*, or None if tag_type not exists

property tag_types
Property of included tag types. None indicates untyped tags included.

Getter Returns a set containing all the tag types of the typed tags

Type set

property tag_values
Property of included tag values. Same tag values will be merged

Getter Returns a set containing all the tag values, both typed and untyped tags

Type set

property tags
Property of all included tags.

Getter Returns a tuple containing all the tags, both typed and untyped

Type tuple

to_list()
Exports tags to a list. Each element is a tuple of (tag value, tag type).

```
>>> tags = PeakList_Tags('untyped_tag1', tag_type1 = 'tag_value1')
>>> tags.to_list()
[('untyped_tag1', None), ('tag_value1', 'tag_type1')]
```

Return type list

to_str()
Exports tags to a string. It can also be used inexplicitly as

```
>>> tags = PeakList_Tags('untyped_tag1', tag_type1 = 'tag_value1')
>>> print(tags)
untyped_tag1, tag_type1:tag_value1
```

Return type str

property typed_tags
Property of included typed tags.

Getter Returns a tuple containing all the typed tags

Type tuple

property untyped_tags
Property of included untyped tags.

Getter Returns a tuple containing all the untyped tags

Type tuple

class dimspy.models.peaklist_tags.**Tag**(*value: Union[int, float, str, dimspy.models.peaklist_tags.Tag], ttype: Optional[str] = None*)

Bases: object

The Tag class.

This class is mainly used in PeakList and PeakMatrix classes for sample filtering.

Parameters

- **value** – Tag value, must be number (int, float), string (ascii, unicode), or Tag object (ignore ttype setting)
- **ttype** – Tag type, must be string or None (untyped), default = None

Single value will be treated as untyped tag:

```
>>> tag = Tag(1)
>>> tag == 1
True
>>> tag = Tag(1, 'batch')
>>> tag == 1
False
```

property ttype

Property of tag type. None indicates untyped tag.

Getter Returns the type of the tag

Setter Set the tag type, must be None or string

Type None, str, unicode

property typed

Property to decide if the tag is typed or untyped.

Getter Returns typed status of the tag

Type bool

property value

Property of tag value.

Getter Returns the value of the tag

Setter Set the tag value, must be number or string

Type int, float, str, unicode

peak_matrix

class dimspy.models.peak_matrix.**PeakMatrix**(*peaklist_ids: Sequence[str], peaklist_tags: Sequence[dimspy.models.peaklist_tags.PeakList_Tags], peaklist_attributes: Sequence[Tuple[str, Any]]*)

Bases: `object`

The PeakMatrix class.

Stores aligned mass spectrometry peaks matrix data. It requires IDs, tags, and attributes from the source peak lists. It uses tags based mask to "hide" the unrelated samples for convenient processing. It utilises the automatically managed flags to "remove" peaks without actually delete them. Therefore the filterings on the peaks are traceable. Normally, PeakMatrix object is created by functions e.g. align_peaks() rather than manual.

Parameters

- **peaklist_ids** – The IDs of the source peak lists
- **peaklist_tags** – The tags (PeakList_Tags) of the source peak lists
- **peaklist_attributes** – The attributes of the source peak lists. Must be a list or tuple in the format of [(attr_name, attr_matrix), ...], where attr_name is name of the attribute, and attr_matrix is the vertically stacked arrtibute values in the shape of samples x peaks. The order of the attributes will be kept in the PeakMatrix. The first two attributes must be "mz" and "intensity".

```
>>> pids = [pl.ID for pl in peaklists]
>>> tags = [pl.tags for pl in peaklists]
>>> attrs = [(attr_name, np.vstack([pl[attr_name] for pl in peaklists]))
↪        for attr_name in peaklists[0].attributes]
>>> pm = PeakMatrix(pids, tags, attrs)
```

Internally the attribute data is stored in OrderedDict as a list of matrix. An attribute matrix can be illustrated as follows, in which the mask and flags are the same for all attributes. The final row "flags" is automatically calculated based on the manually added flags. It decides which peaks are "removed" i.e. unflagged. Particularly, the "–" indicates no peak in that sample can be aligned into the mz value.

attribute: "mz"

mask	peak_1	peak_2	peak_3	...
False	12.7	14.9	21.0	...
True	–	15.1	21.1	
False	12.1	14.7	–	
False	12.9	14.8	20.9	
...				
flag_1	True	False	True	...
flag_2	True	True	False	
flags*	True	False	False	

Warning: Removing a flag may change the overall "flags", and cause the unflagged peaks to be flagged again. As most the processes are applied only on the flagged peaks, these peaks, if the others have gone through such process, may have incorrect values.

In principle, setting a flag attribute should be considered as an irreversible process.

Different from the flags, mask should be considered as a more temporary way to hide the unrelated samples. A masked sample (row) will not be used for processing, but its data is still in the attribute matrix. For this reason, the mask_peakmatrix, unmask_peakmatrix, and unmask_all_peakmatrix statements are provided as a more flexible way to set / unset the mask.

add_flag(*flag_name: str, flag_values: Sequence[bool], flagged_only: bool = True*)
Adds a flag to the peak matrix peaks.

Parameters

- **flag_name** – name of the flag, it must be unique and not equal to "flags"
- **flag_values** – values of the flag. It must have a length of pm.shape[1] if flagged_only = True, or pm.full_shape[1] if flagged_only = False
- **flagged_only** – whether to set the flagged peaks only. Default = True, and the values of the unflagged peaks are set to False

The overall flags property will be automatically recalculated.

attr_matrix(*attr_name: str*, *flagged_only: bool = True*)
Obtains an existing attribute matrix.

Parameters

- **attr_name** – name of the target attribute
- **flagged_only** – whether to return the flagged values only. Default = True

Return type numpy array

attr_mean_vector(*attr_name: str*, *flagged_only: bool = True*)
Obtains the mean array of an existing attribute matrix.

Parameters

- **attr_name** – name of the target attribute
- **flagged_only** – whether to return the mean array of the flagged values only. Default = True

Return type numpy array

Noting that only the "present" peaks will be used for mean values calculation. If the attribute matrix has a string / unicode data type, the values in each column will be concatenated.

property attributes
Property of the attribute names.

Getter returns a tuple including the names of the attribute matrix

Type tuple

drop_flag(*flag_name: str*)
Drops a existing flag from the peak matrix.

Parameters **flag_name** – name of the flag to drop. It must exist and not equal to "flags"

The overall flags property will be automatically recalculated.

extract_peaklist(*peaklist_id: str*)
Extracts one peaklist from the peak matrix.

Parameters **peaklist_id** – ID of the peaklist to extract

Return type PeakList object

Only the "present" peaks will be included in the result peaklist.

extract_peaklists()
Extracts all peaklists from the peak matrix.

Return type list

property flag_names
Property of the flag names.

Getter returns a tuple including the names of the manually set flags

Type tuple

flag_values(*flag_name: str*)
Obtains values of an existing flag.

Parameters **flag_name** – name of the target flag. It must exist and not equal to "flags"

Return type numpy array

property flags
Property of the flags.

Getter returns a deep copy of the flags array

Type numpy array

property fraction
Property of the fraction array.

Getter returns the fraction array, indicating the ratio of present peaks on each mz value

Type numpy array

```
>>> print pm.present
array([3, 4, 2, 3, 3])
>>> print pm.shape[0]
4
>>> print pm.fraction
array([0.75, 1.0, 0.5, 0.75, 0.75])
```

property full_shape
Property of the peak matrix full shape.

Getter returns the full shape of the attribute matrix, i.e., ignore mask and flags

Type tuple

property intensity_matrix
Property of the intensity matrix.

Getter returns the intensity attribute matrix, unmasked and flagged values only

Type numpy array

property intensity_mean_vector
Property of the intensity mean values array.

Getter returns the mean values array of the intensity attribute matrix, unmasked and flagged values only

Type numpy array

is_empty()
Checks whether the peak matrix is empty under the current mask and flags.

Return type bool

property mask
Property of the mask.

Getter returns a deep copy of the mask array

Setter sets the mask array. Provide None to unmask all samples

Type numpy array

mask_tags(**args*, ***kwargs*)
Masks samples with particular tags.

Parameters

- **args** – tags or untyped tag values for masking
- **kwargs** – typed tags for masking
- **override** – whether to override the current mask, default = False

Return type PeakMatrix object (self)

This function will mask samples with ALL the tags. To match ANY of the tags, use cascade form instead.

```
>>> pm.mask_tags('qc', plate = 1)
(will mask all QC samples on plate 1)
>>> pm.mask_tags('qc').mask_tags(plate = 1)
(will mask QC samples and all samples on plate 1)
```

property missing_values
Property of the missing values array.

Getter returns the missing values array, indicating the number of unaligned peaks on each sample

Type numpy array

```
>>> print pm.present_matrix
array([[ True,  True,  True,  True, False],
       [ True,  True, False, False,  True],
       [ True,  True,  True,  True,  True],
       [False,  True, False,  True,  True],])
>>> print pm.missing_values
array([1, 2, 0, 2])
```

property mz_matrix
Property of the mz matrix.

Getter returns the mz attribute matrix, unmasked and flagged values only

Type numpy array

property mz_mean_vector
Property of the mz mean values array.

Getter returns the mean values array of the mz attribute matrix, unmasked and flagged values only

Type numpy array

property occurrence
Property of the occurrence array.

Getter returns the occurrence array, indicating the total number of peaks (including peaks in the same sample) aliged in each mz value. This property is valid only when the *intra_count* attribute matrix is available

Type numpy array

```
>>> print pm.attr_matrix('intra_count')
array([[ 2,  1,  1,  1,  0],
       [ 1,  1,  0,  0,  1],
       [ 1,  3,  1,  2,  1],
       [ 0,  1,  0,  1,  1],])
>>> print pm.occurrence
array([ 4,  6,  2,  4,  3])
```

property peaklist_ids
Property of the source peaklist IDs.

Getter returns a tuple including the IDs of the source peaklists

Type tuple

property peaklist_tag_types
Property of the source peaklist tag types.

Getter returns a tuple including the types of the typed tags of the source peaklists

Type set

property peaklist_tag_values
Property of the source peaklist tag values.

Getter returns a tuple including the values of the source peaklists tags, both typed and untyped

Type set

property peaklist_tags
Property of the source peaklist tags.

Getter returns a tuple including the Peaklist_Tags objects of the source peaklists

Type tuple

property present
Property of the present array.

Getter returns the present array, indicating how many peaks are aligned in each mz value

Type numpy array

property present_matrix
Property of the present matrix.

Getter returns the present matrix, indicating whether a sample has peak(s) aligned in each mz value

Type numpy array

```
>>> print pm.present_matrix
array([[ True,  True,  True,  True, False],
       [ True,  True, False, False,  True],
       [ True,  True,  True,  True,  True],
       [False,  True, False,  True,  True],])
>>> print pm.present
array([3, 4, 2, 3, 3])
```

property (*prop_name: str*, *flagged_only: bool = True*)
Obtains an existing attribute matrix.

Parameters

- **prop_name** – name of the target property. Valid properties include 'present', 'present_matrix', 'fraction', 'missing_values', 'occurrence', and 'purity'
- **flagged_only** – whether to return the flagged values only. Default = True

Return type numpy array

property purity
Property of the purity level array.

Getter returns the purity array, indicating the ratio of only one peak in each sample being aligned in each mz value. This property is valid only when the *intra_count* attribute matrix is available

Type numpy array

```
>>> print pm.attr_matrix('intra_count')
array([[ 2,  1,  1,  1,  0],
       [ 1,  1,  0,  0,  1],
       [ 1,  3,  1,  2,  1],
       [ 0,  1,  0,  1,  1],])
>>> print pm.purity
array([ 0.667,  0.75,  1.0,  0.667,  1.0])
```

remove_empty_peaks()
Removes empty peaks from the peak matrix.

Empty peaks are peaks with not valid m/z or intensity value over the samples. They may occur after removing an entire sample from the peak matrix, e.g., remove the blank samples in the blank filter.

Return type PeakMatrix object (self)

remove_peaks (*peak_ids*, *flagged_only: bool = True*)
Removes peaks from the peak matrix.

Parameters

- **peak_ids** – the indices of the peaks to remove
- **flagged_only** – whether the indices are for flagged peaks or all peaks. Default = True

Return type PeakMatrix object (self)

remove_samples (*sample_ids*, *masked_only: bool = True*)
Removes samples from the peak matrix.

Parameters

- **sample_ids** – the indices of the samples to remove
- **masked_only** – whether the indices are for unmasked samples or all samples. Default = True

Return type PeakMatrix object (self)

rsd (**args*, ***kwargs*)
Calculates relative standard deviation (RSD) array.

Parameters

- **args** – tags or untyped tag values for RSD calculation, no value = calculate over all samples
- **kwargs** – typed tags for RSD calculation, no value = calculate over all samples
- **on_attr** – calculate RSD on given attribute. Default = "intensity"
- **flagged_only** – whether to calculate on flagged peaks only. Default = True

Type numpy array

The RSD is calculated as:

```
>>> rsd = std(pm.intensity_matrix, axis = 0, ddof = 1) / mean(pm.intensity_
↪matrix, axis = 0) * 100
```

Noting that the means delta degrees of freedom (ddof) is set to 1 for standard deviation calculation. Moreover, only the "present" peaks will be used for calculation. If a column has less than 2 peaks, the corresponding rsd value will be set to np.nan.

property shape

Property of the peak matrix shape.

Getter returns the shape of the attribute matrix

Type tuple

tags_of (*tag_type: Optional[str] = None*)

Obtains tags of the peaklist_tags with particular tag type.

Parameters **tag_type** – the type of the returning tags. Provide None to obtain untyped tags

Return type tuple

to_peaklist (*ID: str*)

Averages the peak matrix into a single peaklist.

Parameters **ID** – ID of the merged peaklist

Return type PeakList object

Only the "present" peaks will be included in the result peaklist. The new peaklist will only contain the following attributes: mz, intensity, present, fraction, rsd, occurence, and purity.

Use unmask statement to calculate the peaklist for a particular group of samples:

```
>>> with unmask_peakmatrix(pm, 'Sample') as m: pkl = m.to_peaklist('averaged_
↪peaklist')
```

Or use mask statement to exclude a particular group of samples:

```
>>> with mask_peakmatrix(pm, 'QC') as m: pkl = m.to_peaklist('averaged_
↪peaklist')
```

to_str (*attr_name: str = 'intensity', delimiter: str = '\t', samples_in_rows: bool = True, comprehensive: bool = True, rsd_tags: Sequence = ()*)

Exports the peak matrix to a string.

Parameters

- **attr_name** – name of the attribute matrix for exporting. Default = 'intensity'
- **delimiter** – delimiter to separate the matrix. Default = ' ', i.e., TSV format
- **samples_in_rows** – whether or not the samples are stored in rows. Default = True

- **comprehensive** – whether to include comprehensive info, e.g., mask, flags, present, rsd etc. Default = True
- **rsd_tags** – peaklist tags for RSD calculation. Default = (), indicating only the overall RSD is included

Return type str

unmask_tags(**args*, ***kwargs*)

Unmasks samples with particular tags.

Parameters

- **args** – tags or untyped tag values for unmasking
- **kwargs** – typed tags for unmasking
- **override** – whether to override the current mask, default = False

Return type PeakMatrix object (self)

This function will unmask samples with ALL the tags. To unmask ANY of the tags, use cascade form instead.

```
>>> pm.mask = [True] * pm.full_shape[0]
>>> pm.unmask_tags('qc', plate = 1)
(will unmask all QC samples on plate 1)
>>> pm.unmask_tags('qc').unmask_tags(plate = 1)
(will unmask QC samples and all samples on plate 1)
```

class dimspy.models.peak_matrix.**mask_all_peakmatrix**(*pm: dimspy.models.peak_matrix.PeakMatrix*)

Bases: object

The mask_all_peakmatrix statement.

Temporary mask all the peak matrix samples. Within the statement the samples can be motified or removed. After leaving the statement the original mask will be recoverd.

Parameters **pm** – the target peak matrix

Return type PeakMatrix object

```
>>> print pm.peaklist_ids
('sample_1', 'sample_2', 'qc_1', 'sample_3', 'sample_4', 'qc_2')
>>> with mask_all_peakmatrix(pm) as m: print m.peaklist_ids
()
>>> print pm.peaklist_ids
('sample_1', 'sample_2', 'qc_1', 'sample_3', 'sample_4', 'qc_2')
```

class dimspy.models.peak_matrix.**mask_peakmatrix**(*pm: dimspy.models.peak_matrix.PeakMatrix*, **args*, ***kwargs*)

Bases: object

The mask_peakmatrix statement.

Temporary mask the peak matrix with particular tags. Within the statement the samples can be motified or removed. After leaving the statement the original mask will be recoverd.

Parameters

- **pm** – the target peak matrix
- **override** – whether to override the current mask, default = True

- **args** – target tag values, both typed and untyped
- **kwargs** – target typed tag types and values

Return type PeakMatrix object

```
>>> print pm.peaklist_ids
('sample_1', 'sample_2', 'qc_1', 'sample_3', 'sample_4', 'qc_2')
>>> with mask_peakmatrix(pm., 'qc') as m: print m.peaklist_ids
('sample_1', 'sample_2', 'sample_3', 'sample_4')
>>> print pm.peaklist_ids
('sample_1', 'sample_2', 'qc_1', 'sample_3', 'sample_4', 'qc_2')
```

class dimspy.models.peak_matrix.**unmask_all_peakmatrix**(*pm: dimspy.models.peak_matrix.PeakMatrix*)

Bases: object

The unmask_all_peakmatrix statement.

Temporary unmask all the peak matrix samples. Within the statement the samples can be motified or removed. After leaving the statement the original mask will be recoverd.

Parameters **pm** – the target peak matrix

Return type PeakMatrix object

```
>>> print pm.peaklist_ids
('sample_1', 'sample_2', 'qc_1', 'sample_3', 'sample_4', 'qc_2')
>>> with unmask_all_peakmatrix(pm) as m: print m.peaklist_ids
('sample_1', 'sample_2', 'qc_1', 'sample_3', 'sample_4', 'qc_2')
>>> print pm.peaklist_ids
('sample_1', 'sample_2', 'qc_1', 'sample_3', 'sample_4', 'qc_2')
```

class dimspy.models.peak_matrix.**unmask_peakmatrix**(*pm: dimspy.models.peak_matrix.PeakMatrix*, **args*, ***kwargs*)

Bases: object

The unmask_peakmatrix statement.

Temporary unmask the peak matrix with particular tags. Within the statement the samples can be motified or removed. After leaving the statement the original mask will be recoverd.

Parameters

- **pm** – the target peak matrix
- **override** – whether to override the current mask, default = True
- **args** – target tag values, both typed and untyped
- **kwargs** – target typed tag types and values

Return type PeakMatrix object

```
>>> print pm.peaklist_ids
('sample_1', 'sample_2', 'qc_1', 'sample_3', 'sample_4', 'qc_2')
>>> with unmask_peakmatrix(pm, 'qc') as m: print m.peaklist_ids
('qc_1', 'qc_2') # no need to set pm.mask to True
>>> print pm.peaklist_ids
('sample_1', 'sample_2', 'qc_1', 'sample_3', 'sample_4', 'qc_2')
```

1.2.4 portals

mzml_portal

class dimspy.portals.mzml_portal.**Mzml**(*filename: Union[str, _io.BytesIO], **kwargs*)
Bases: object

mzML portal

headers() → collections.OrderedDict
Get all unique header or filter strings and associated scan ids. :return: Dictionary

scan_ids() → collections.OrderedDict
Get all scan ids and associated headers or filter strings. :return: Dictionary

peaklist(*scan_id*, *function_noise='median'*) → dimspy.models.peaklist.PeakList
Create a peaklist object for a specific scan id. :param scan_id: Scan id :param function_noise: Function to calculate the noise from each scan. The following options are available:

- **median** - the median of all peak intensities within a given scan is used as the noise value.
- **mean** - the unweighted mean average of all peak intensities within a given scan is used as the noise value.
- **mad (Mean Absolute Deviation)** - the noise value is set as the mean of the absolute differences between peak intensities and the mean peak intensity (calculated across all peak intensities within a given scan).

Returns PeakList object

peaklists(*scan_ids*, *function_noise='median'*) → Sequence[dimspy.models.peaklist.PeakList]
Create a list of peaklist objects for each scan id in the list. :param scan_ids: List of scan ids

Parameters **function_noise** – Function to calculate the noise from each scan. The following options are available:

- **median** - the median of all peak intensities within a given scan is used as the noise value.
- **mean** - the unweighted mean average of all peak intensities within a given scan is used as the noise value.
- **mad (Mean Absolute Deviation)** - the noise value is set as the mean of the absolute differences between peak intensities and the mean peak intensity (calculated across all peak intensities within a given scan).
- **noise_packets** - the noise value is calculated using the proprietary algorithms contained in Thermo Fisher Scientific's msFileReader library. This option should only be applied when you are processing .RAW files.

Returns List of PeakList objects

tics() → collections.OrderedDict
Get all TIC values and associated scan ids :return: Dictionary

ion_injection_times() → collections.OrderedDict
Get all ion injection time values and associated scan ids :return: Dictionary

scan_dependents() → list
Get a nested list of scan id pairs. Each pair represents a fragementation event. :return: List

close()
Close the reader/file object :return: None

thermo_raw_portal

dimspy.portals.thermo_raw_portal.**mz_range_from_header**(*h: str*) → list
Extract the m/z range from a header or filterstring

Parameters **h** – str

Returns Sequence[float, float]

class dimspy.portals.thermo_raw_portal.**ThermoRaw**(*filename*)
Bases: object

ThermoRaw portal

headers() → collections.OrderedDict
Get all unique header or filter strings and associated scan ids. :return: Dictionary

scan_ids() → collections.OrderedDict
Get all scan ids and associated headers or filter strings. :return: Dictionary

peaklist(*scan_id*, *function_noise='noise_packets'*) → dimspy.models.peaklist.PeakList
Create a peaklist object for a specific scan id. :param scan_id: Scan id :param function_noise: Function to calculate the noise from each scan. The following options are available:

- **median** - the median of all peak intensities within a given scan is used as the noise value.
- **mean** - the unweighted mean average of all peak intensities within a given scan is used as the noise value.
- **mad (Mean Absolute Deviation)** - the noise value is set as the mean of the absolute differences between peak intensities and the mean peak intensity (calculated across all peak intensities within a given scan).
- **noise_packets** - the noise value is calculated using the proprietary algorithms contained in Thermo Fisher Scientific's msFileReader library. This option should only be applied when you are processing .RAW files.

Returns PeakList object

peaklists(*scan_ids*, *function_noise='noise_packets'*) → Sequence[dimspy.models.peaklist.PeakList]
Create a list of peaklist objects for each scan id in the list. :param scan_ids: List of scan ids

Parameters **function_noise** – Function to calculate the noise from each scan. The following options are available:

- **median** - the median of all peak intensities within a given scan is used as the noise value.
- **mean** - the unweighted mean average of all peak intensities within a given scan is used as the noise value.
- **mad (Mean Absolute Deviation)** - the noise value is set as the mean of the absolute differences between peak intensities and the mean peak intensity (calculated across all peak intensities within a given scan).
- **noise_packets** - the noise value is calculated using the proprietary algorithms contained in Thermo Fisher Scientific's msFileReader library. This option should only be applied when you are processing .RAW files.

Returns List of PeakList objects

tics() → collections.OrderedDict
Get all TIC values and associated scan ids :return: Dictionary

ion_injection_times() → collections.OrderedDict
Get all TIC values and associated scan ids :return: Dictionary

scan_dependents() → list
Get a nested list of scan id pairs. Each pair represents a fragementation event. :return: List

close()
Close the reader/file object :return: None

txt_portal

dimspy.portals.txt_portal.**save_peaklist_as_txt**(*pkl: dimspy.models.peaklist.PeakList*, *filename: str*, **args*, ***kwargs*)

Saves a peaklist object to a plain text file.

Parameters

- **pkl** – the target peaklist object
- **filename** – path to a new text file
- **args** – arguments to be passed to PeakList.to_str
- **kwargs** – keyword arguments to be passed to PeakList.to_str

dimspy.portals.txt_portal.**load_peaklist_from_txt**(*filename: str*, *ID: any*, *delimiter: str = ', '*, *flag_names: str = 'auto'*, *has_flag_col: bool = True*)

Loads a peaklist from plain text file.

Parameters

- **filename** – Path to an exiting text-based peaklist file
- **ID** – ID of the peaklist
- **delimiter** – Delimiter of the text lines. Default = ',', i.e., CSV format
- **flag_names** – Names of the flag attributes. Default = 'auto', indicating all the attribute names ends with "_flag" will be treated as flag attibute. Provide None to indicate no flag attributes
- **has_flag_col** – Whether the text file contains the overall "flags" column. If True, it's values will be discarded. The overall flags of the new peaklist will be calculated automatically. Default = True

Return type PeakList object

dimspy.portals.txt_portal.**save_peak_matrix_as_txt**(*pm: dimspy.models.peak_matrix.PeakMatrix*, *filename: str*, **args*, ***kwargs*)

Saves a peak matrix in plain text file.

Parameters

- **pm** – The target peak matrix object
- **filename** – Path to a new text file

- **args** – Arguments to be passed to PeakMatrix.to_str
- **kwargs** – Keyword arguments to be passed to PeakMatrix.to_str

dimspy.portals.txt_portal.**load_peak_matrix_from_txt**(*filename: str*, *delimiter: str = '\t'*, *samples_in_rows: bool = True*, *comprehensive: str = 'auto'*)

Loads a peak matrix from plain text file.

Parameters

- **filename** – Path to an exiting text-based peak matrix file
- **delimiter** – Delimiter of the text lines. Default = ' ', i.e., TSV format
- **samples_in_rows** – Whether or not the samples are stored in rows. Default = True
- **comprehensive** – Whether the input is a 'comprehensive' or 'simple' version of the matrix. Default = 'auto', i.e., auto detect

Return type PeakMatrix object

hdf5_portal

dimspy.portals.hdf5_portal.**save_peaklists_as_hdf5**(*pkls: Sequence[dimspy.models.peaklist.PeakList]*, *filename: str*, *compatibility_mode: bool = False*)

Saves multiple peaklists in a HDF5 file.

Parameters

- **pkls** – The target list of peaklist objects
- **filename** – Path to a new HDF5 file
- **compatibility_mode** – Change mode to read previous DIMSpy v1.* based HDF5 file

To incorporate with different dtypes in the attribute matrix, this portal converts all the arribute values into fix-length strings for HDF5 data tables storage. The order of the peaklists will be retained.

dimspy.portals.hdf5_portal.**load_peaklists_from_hdf5**(*filename: str*, *compatibility_mode: bool = False*)

Loads a list of peaklist objects from a HDF5 file.

Parameters

- **filename** – Path to a HDF5 file
- **compatibility_mode** – Change mode to read previous DIMSpy v1.* based HDF5 file

Return type Sequence[*PeakList*]

The values in HDF5 data tables are automatically converted to their original dtypes before loading in the peaklist.

dimspy.portals.hdf5_portal.**save_peak_matrix_as_hdf5**(*pm: dimspy.models.peak_matrix.PeakMatrix*, *filename: str*, *compatibility_mode: bool = False*)

Saves a peak matrix object to a HDF5 file.

Parameters

- **pm** – The target peak matrix object

- **filename** – Path to a new HDF5 file

The order of the attributes and flags will be retained.

dimspy.portals.hdf5_portal.**load_peak_matrix_from_hdf5**(*filename: str, compatibility_mode: bool = False*)

Loads a peak matrix from a HDF5 file.

Parameters **filename** – Path to an existing HDF5 file

Return type PeakMatrix object

paths

dimspy.portals.paths.**sort_ms_files_by_timestamp**(*ps*)

Sort a set directory of .mzml or .raw files

Parameters **ps** – List of paths

:return List

dimspy.portals.paths.**validate_and_sort_paths**(*source*, *tsv*)

Validate and sort a set (i.e. directory or hdf5 file) of .mzml or .raw files.

Parameters

- **tsv** – Path to tab-separated file
- **source** – Path to a Path to the .hdf5 file to read from.

Returns List

1.2.5 process

peak_alignment

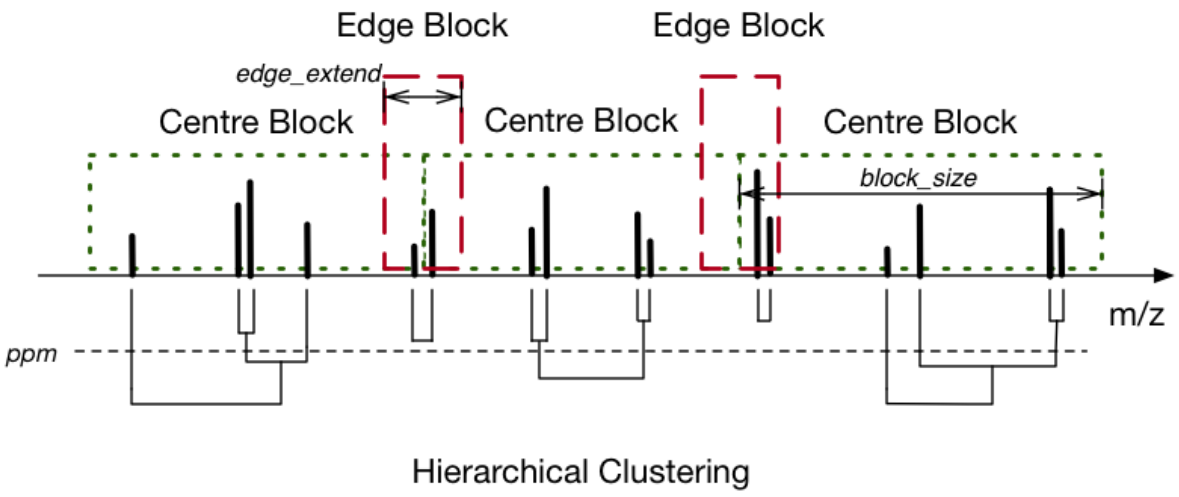
dimspy.process.peak_alignment.**align_peaks**(*peaks: Sequence[dimspy.models.peaklist.PeakList], ppm: float = 2.0, block_size: int = 5000, fixed_block: bool = True, edge_extend: Union[int, float] = 10, ncpus: Optional[int] = None*)

Cluster and align peaklists into a peak matrix.

Parameters

- **peaks** – List of peaklists for alignment
- **ppm** – The hierarchical clustering cutting height, i.e., ppm range for each aligned mz value. Default = 2.0
- **block_size** – number peaks in each centre clustering block. This can be a exact or approximate number depends on the fixed_block parameter. Default = 5000
- **fixed_block** – Whether the blocks contain fixed number of peaks. Default = True
- **edge_extend** – Ppm range for the edge blocks. Default = 10
- **ncpus** – Number of CPUs for parallel clustering. Default = None, indicating using as many as possible

Return type PeakMatrix object



This function uses hierarchical clustering to align the mz values of the input peaklists. The alignment "width" is decided by the parameter of ppm. Due to a large number of peaks, this function splits them into blocks with fixed or approximate length, and clusters in a parallel manner on multiple CPUs. When running, the edge blocks are clustered first to prevent separating the same peak into two adjacent centre blocks. The size of the edge blocks is decided by edge_extend. The clustering of centre blocks is conducted afterwards.

After merging the clustering results, all the attributes (mz, intensity, snr, etc.) are aligned into matrix accordingly. If multiple peaks from the same sample are clustered into one mz value, their attributes are averaged (for real value attributes e.g. mz and intensity) or concatenated (string, unicode, or bool attributes). The flag attributes are ignored. The number of these overlapping peaks is recorded in a new intra_count attribute matrix.

peak_filters

dimspy.process.peak_filters.**filter_attr**(*pl: dimspy.models.peaklist.PeakList*, *attr_name: str*, *max_threshold: Union[int, float, None] = None*, *min_threshold: [<class 'int'>, <class 'float'>, None] = None*, *flag_name: Optional[str] = None*, *flag_index: Optional[int] = None*)

Peaklist attribute values filter.

Parameters

- **pl** – The target peaklist
- **attr_name** – Name of the target attribute
- **max_threshold** – Maximum threshold. A peak will be unflagged if the value of it's attr_name is larger than the threshold. Default = None, indicating no threshold
- **min_threshold** – Minimum threshold. A peak will be unflagged if the value of it's attr_name is smaller than the threshold. Default = None, indicating no threshold
- **flag_name** – Name of the new flag attribute. Default = None, indicating using attr_name + '_flag'
- **flag_index** – Index of the new flag to be inserted into the peaklist. Default = None

Return type PeakList object

This filter accepts real value attributes only.

dimspy.process.peak_filters.**filter_ringing**(*pl: dimspy.models.peaklist.PeakList, threshold: float, bin_size: Union[int, float] = 1.0, flag_name: str = 'ringing_flag', flag_index: Optional[int] = None*)

Peaklist ringing filter.

Parameters

- **pl** – The target peaklist
- **threshold** – Intensity threshold ratio
- **bin_size** – size of the mz chunk for intensity filtering. Default = 1.0 ppm
- **flag_name** – Name of the new flag attribute. Default = 'ringing_flag'
- **flag_index** – Index of the new flag to be inserted into the peaklist. Default = None

Return type PeakList object

This filter will split the mz values into bin_size chunks, and search the highest intensity value for each chunk. All other peaks, if it's intensity is smaller than threshold x the highest intensity in that chunk, will be unflagged.

dimspy.process.peak_filters.**filter_mz_ranges**(*pl: dimspy.models.peaklist.PeakList, mz_ranges: Sequence[Tuple[float, float]], flag_name: str = 'mz_ranges_flag', flagged_only: bool = False, flag_index: Optional[int] = None*)

Peaklist mz range filter.

Parameters

- **pl** – The target peaklist
- **mz_ranges** – The mz ranges to remove. Must be in the format of [(mz_min1, mz_max2), (mz_min2, mz_max2), . . .]
- **flag_name** – Name of the new flag attribute. Default = 'mz_range_remove_flag'
- **flag_index** – Index of the new flag to be inserted into the peaklist. Default = None

Return type *PeakList*

This filter will remove all the peaks whose mz values are within any of the ranges in the mz_remove_rngs.

dimspy.process.peak_filters.**filter_rsd**(*pm: dimspy.models.peak_matrix.PeakMatrix, rsd_threshold: Union[int, float], qc_tag: Any, on_attr: str = 'intensity', flag_name: str = 'rsd_flag'*)

PeakMatrix RSD filter.

Parameters

- **pm** – The target peak matrix
- **rsd_threshold** – Threshold of the RSD of the QC samples
- **qc_tag** – Tag (label) to unmask qc samples
- **on_attr** – Calculate RSD on given attribute. Default = "intensity"
- **flag_name** – Name of the new flag. Default = 'rsd_flag'

Return type *PeakMatrix*

This filter will calculate the RSD values of the QC samples. A peak with a QC RSD value larger than the threshold will be unflagged.

dimspy.process.peak_filters.**filter_fraction**(*pm: dimspy.models.peak_matrix.PeakMatrix*, *fraction_threshold: float*, *within_classes: bool = False*, *class_tag_type: Any = None*, *flag_name: str = 'fraction_flag'*)

PeakMatrix fraction filter.

Parameters

- **pm** – The target peak matrix
- **fraction_threshold** – Threshold of the sample fractions
- **within_classes** – Whether to calculate the fraction array within each class. Default = False
- **class_tag_type** – Tag type to unmask samples within the same class (e.g. "classLabel"). Default = None
- **flag_name** – Name of the new flag. Default = 'fraction_flag'

Return type PeakMatrix object

This filter will calculate the fraction array over all samples or within each class (based on class_tag_type). The peaks with a fraction value smaller than the threshold will be unflagged.

dimspy.process.peak_filters.**filter_blank_peaks**(*pm: dimspy.models.peak_matrix.PeakMatrix*, *blank_tag: Any*, *fraction_threshold: Union[int, float] = 1*, *fold_threshold: Union[int, float] = 1*, *method: str = 'mean'*, *rm_blanks: bool = True*, *flag_name: str = 'blank_flag'*)

PeakMatrix blank filter.

Parameters

- **pm** – The target peak matrix
- **blank_tag** – Tag (label) to mask blank samples. e.g Tag("blank", "classLabel")
- **fraction_threshold** – Threshold of the sample fractions. Default = 1
- **fold_threshold** – Threshold of the blank sample intensity folds. Default = 1
- **method** – Method to calculate blank sample intensity array. Valid values include 'mean', 'median', and 'max'. Default = 'mean'
- **rm_blanks** – Whether to remove (not mask) blank samples after filtering
- **flag_name** – Name of the new flag. Default = 'blank_flag'

Return type PeakMatrix object

This filter will calculate the intensity array of the blanks using the "method", and compare with the intensities of the other samples. If fraction_threshold% of the intensity values of a peak are smaller than the blank intensities x fold_threshold, this peak will be unflagged.

scan_processing

dimspy.process.replicate_processing.**remove_edges**(*pls_sd: Dict*)

Removes overlapping m/z regions of adjacent (SIM) windows / scan events.

Parameters **pls_sd** – List of peaklist objects

Returns List of peaklist objects

dimspy.process.replicate_processing.**read_scans**(*fn: str*, *function_noise: str*, *min_scans: int = 1*, *filter_scan_events: Dict = None*)

Read, filter, group and sort scans based on the header / filter string Helper function for 'process_scans (tools module)'

Parameters

- **fn** – Path to the .mzml or .raw file
- **function_noise** – Function to calculate the noise from each scan. The following options are available:
  - **median** - the median of all peak intensities within a given scan is used as the noise value.
  - **mean** - the unweighted mean average of all peak intensities within a given scan is used as the noise value.
  - **mad (Mean Absolute Deviation)** - the noise value is set as the mean of the absolute differences between peak intensities and the mean peak intensity (calculated across all peak intensities within a given scan).
  - **noise_packets** - the noise value is calculated using the proprietary algorithms contained in Thermo Fisher Scientific's msFileReader library. This option should only be applied when you are processing .RAW files.
- **min_scans** – Minimum number of scans required for each *m/z* window or event within a raw/mzML data file.
- **filter_scan_events** – Include or exclude specific scan events, by default all ALL scan events will be included. To include or exclude specific scan events use the following format of a dictionary.

```
>>> {"include":[[100, 300, "sim"]]} or {"include":[[100, 1000,
↪"full"]]}
```

Returns List of peaklist objects

dimspy.process.replicate_processing.**average_replicate_scans**(*name: str*, *pls: Sequence[dimspy.models.peaklist.PeakList]*, *ppm: float = 2.0*, *min_fraction: float = 0.8*, *rsd_thres: float = 30.0*, *rsd_on: str = 'intensity'*, *block_size: int = 5000*, *ncpus: int = None*)

Align, filter and average replicate scans/peaklist Helper function for 'process_scans (tools module)'

Parameters

- **name** – Name average peaklist

- **pls** – List of peaklists
- **ppm** – Maximum tolerated m/z deviation in parts per million.
- **min_fraction** – A numerical value from 0 to 1 that specifies the minimum proportion of scans a given mass spectral peak must be detected in, in order for it to be kept in the output peaklist. Here, scans refers to replicates of the same scan event type, i.e. if set to 0.33, then a peak would need to be detected in at least 1 of the 3 replicates of a given scan event type.
- **rsd_thres** – Relative standard deviation threshold - A numerical value equal-to or greater-than 0. If greater than 0, then peaks whose intensity values have a percent relative standard deviation (otherwise termed the percent coefficient of variation) greater-than this value are excluded from the output peaklist.
- **rsd_on** – Intensity or SNR
- **block_size** – Number peaks in each centre clustering block.
- **ncpus** – Number of CPUs for parallel clustering. Default = None, indicating using all CPUs that are available

Returns List of peaklists

dimspy.process.replicate_processing.**average_replicate_peaklists**(*pls: Sequence[dimspy.models.peaklist.PeakLis ppm: float, min_peaks: int, rsd_thres: float = None, block_size: int = 5000, ncpus: int = None*)

Align, filter and average replicate peaklists. Helper function for 'replicate_filter (tools module)'

Parameters

- **pls** – List of peaklists
- **ppm** – Maximum tolerated m/z deviation in parts per million.
- **min_peaks** – Minimum number of technical replicates (i.e. peaklists) a peak has to be present in.
- **rsd_thres** – Relative standard deviation threshold - A numerical value equal-to or greater-than 0. If greater than 0, then peaks whose intensity values have a percent relative standard deviation (otherwise termed the percent coefficient of variation) greater-than this value are excluded from the output peaklist.
- **block_size** – Number peaks in each centre clustering block.
- **ncpus** – Number of CPUs for parallel clustering. Default = None, indicating using all CPUs that are available

Returns List of peaklists

dimspy.process.replicate_processing.**join_peaklists**(*name: str, pls: Sequence[dimspy.models.peaklist.PeakList]*)

Join/Merge peaklists (i.e. windows) with different m/z ranges. Helper function for 'process_scans (tools module)'

Parameters

- **name** – Name newly created joined/merged peaklist

- **pls** – List of peaklists

Returns Peaklist

1.3 Command Line Interface

```
$ dimspy --help

Executing dimspy version 2.0.0.
usage: __main__.py [-h]
                   {process-scans,replicate-filter,align-samples,blank-filter,sample-
↪filter,remove-samples,mv-sample-filter,merge-peaklists,get-peaklists,get-average-
↪peaklist,hdf5-pm-to-txt,hdf5-pls-to-txt,create-sample-list,unzip,licenses}
                   ...

Python package to process DIMS data

positional arguments:
  {process-scans,replicate-filter,align-samples,blank-filter,sample-filter,remove-
↪samples,mv-sample-filter,merge-peaklists,get-peaklists,get-average-peaklist,hdf5-pm-
↪to-txt,hdf5-pls-to-txt,create-sample-list,unzip,licenses}
    process-scans       Process scans and/or stitch SIM windows.
    replicate-filter    Filter irreproducible peaks from technical replicate
                        peaklists.
    align-samples       Align peaklists across samples.
    blank-filter        Filter peaks across samples that are present in the
                        blank samples.
    sample-filter       Filter peaks based on certain reproducibility and
                        sample class criteria.
    remove-samples      Remove sample(s) from a peak matrix object or list of
                        peaklist objects.
    mv-sample-filter    Filter samples based on the percentage of missing
                        values.
    merge-peaklists     Merge peaklists from multiple lists of peaklist or
                        peak matrix objects.
    get-peaklists       Get peaklists from a peak matrix object.
    get-average-peaklist
                        Get an average peaklist from a peak matrix object.
    hdf5-pm-to-txt      Write HDF5 output (peak matrix) to text format.
    hdf5-pls-to-txt     Write HDF5 output (peak lists) to text format.
    create-sample-list  Create a sample list from a peak matrix object or list
                        of peaklist objects.
    unzip               Extract files from zip file
    licenses            Show licenses DIMSpy and RawFileReader

optional arguments:
  -h, --help            show this help message and exit
```

```
$ dimspy process-scans --help

Executing dimspy version 2.0.0b1.
usage: __main__.py process-scans [-h] -i source -o OUTPUT [-l FILELIST] -m
                                 {median,mean,mad,noise_packets} -s
                                 SNR_THRESHOLD [-p PPM] [-n MIN_SCANS]
                                 [-a MIN_FRACTION] [-d RSD_THRESHOLD] [-k]
                                 [-r RINGING_THRESHOLD]
```

(continues on next page)

(continued from previous page)

```
                                [-e start end scan_type]
                                [-x start end scan_type] [-z start end]
                                [-u REPORT] [-b BLOCK_SIZE] [-c NCPUS]

optional arguments:
  -h, --help            show this help message and exit
  -i source, --input source
                        Directory (*.raw, *.mzml or tab-delimited peaklist
                        files), single *.mzml/*.raw file or zip archive
                        (*.mzml only)
  -o OUTPUT, --output OUTPUT
                        HDF5 file to save the peaklist objects to.
  -l FILELIST, --filelist FILELIST
                        Tab-delimited file that include the name of the data
                        files (*.raw or *.mzml) and meta data. Column names:
                        filename, replicate, batch, injectionOrder,
                        classLabel.
  -m {median,mean,mad,noise_packets}, --function-noise {median,mean,mad,noise_packets}
                        Select function to calculate noise.
  -s SNR_THRESHOLD, --snr-threshold SNR_THRESHOLD
                        Signal-to-noise threshold
  -p PPM, --ppm PPM     Mass tolerance in Parts per million to group peaks
                        across scans / mass spectra.
  -n MIN_SCANS, --min_scans MIN_SCANS
                        Minimum number of scans required for each m/z range or
                        event.
  -a MIN_FRACTION, --min-fraction MIN_FRACTION
                        Minimum fraction a peak has to be present. Use 0.0 to
                        not apply this filter.
  -d RSD_THRESHOLD, --rsd-threshold RSD_THRESHOLD
                        Maximum threshold - relative standard deviation
                        (Calculated for peaks that have been measured across a
                        minimum of two scans).
  -k, --skip-stitching  Skip the step where (SIM) windows are 'stitched' or
                        'joined' together. Individual peaklists are generated
                        for each window.
  -r RINGING_THRESHOLD, --ringing-threshold RINGING_THRESHOLD
                        Ringing
  -e start end scan_type, --include-scan-events start end scan_type
                        Scan events to select. E.g. 100.0 200.0 sim or 50.0
                        1000.0 full
  -x start end scan_type, --exclude-scan-events start end scan_type
                        Scan events to select. E.g. 100.0 200.0 sim or 50.0
                        1000.0 full
  -z start end, --remove-mz-range start end
                        M/z range(s) to remove. E.g. 100.0 102.0 or 140.0
                        145.0.
  -u REPORT, --report REPORT
                        Summary/Report of processed mass spectra
  -b BLOCK_SIZE, --block-size BLOCK_SIZE
                        The size of each block of peaks to perform clustering
                        on.
  -c NCPUS, --ncpus NCPUS
                        Number of central processing units (CPUs).
```

1.4 Credits

DIMSpy was originally written by Ralf Weber and Albert Zhou and has been developed with the help of many others. Thanks to everyone who has improved DIMSpy contributing code, features, bug reports (and fixes), and documentation.

1.4.1 Developers & Contributors

- Ralf J. M. Weber (r.j.weber@bham.ac.uk) - University of Birmingham (UK)
- Jiarui (Albert) Zhou (j.zhou.3@bham.ac.uk) - University of Birmingham (UK), HIT Shenzhen (China)
- Thomas N. Lawson (t.n.lawson@bham.ac.uk) - University of Birmingham (UK)
- Martin R. Jones (martin.jones@eawag.ch) - Eawag (Switzerland)

1.4.2 Funding

DIMSpy acknowledges support from the following funders:

- BBSRC, grant number BB/M019985/1
- European Commission's H2020 programme, grant agreement number 654241
- Wellcome Trust, grant number 202952/Z/16/Z

1.5 Bugs and Issues

Please report any bugs that you find here. Or fork the repository on GitHub and create a pull request (PR). We welcome all contributions, and we will help you to make the PR if you are new to *git*.

1.6 Changelog

All notable changes to this project will be documented here. For more details changes please refer to github commit history

1.6.1 DIMSpy v2.0.0

Release date: 26 April 2020

- First stable Python 3 only release
- Refactor and improve HDF5 portal to save peaklists and/or peak matrices
- Add compatibility for previous HDF5 files (python 2 version of DIMSpy)
- Improve filelist handling
- mzML or raw files are ordered by timestamp if no filelist is provided (i.e. process_scans)
- Fix warnings (NaturalNameWarning, ResourceWarning, DeprecationWarning)
- Fix 'blank filter' bug (missing and/or zero values are excluded)

- Improve sub setting / filtering of scan events
- Optimise imports
- Increase coverage tests
- Improve documentation (Read the Docs), including docstrings

1.6.2 DIMSpy v1.4.0

Release date: 2 October 2019

- Final Python 2 release

1.6.3 DIMSpy v1.3.0

Release date: 26 November 2018

1.6.4 DIMSpy v1.2.0

Release date: 29 May 2018

1.6.5 DIMSpy v1.1.0

Release date: 19 February 2018

1.6.6 DIMSpy v1.0.0

Release date: 10 December 2017

1.6.7 DIMSpy v0.1.0 (pre-release)

Release date: 11 July 2017

1.7 Citation

To cite DIMSpy please use the following publication.

Check Zenodo for citing more up-to-date versions of DIMSpy if not listed here.

DIMSpy v2.0.0

> Ralf J. M. Weber & Jiarui Zhou. (2020, April 24). DIMSpy: Python package for processing direct-infusion mass spectrometry-based metabolomics and lipidomics data (Version v2.0.0). Zenodo. http://doi.org/10.5281/zenodo.3764169

BibTeX

```
@software{ralf_j_m_weber_2020_3764169,
  author       = {Ralf J. M. Weber and
                  Jiarui Zhou},
  title        = {{DIMSpy: Python package for processing direct-
                   infusion mass spectrometry-based metabolomics and
                   lipidomics data}},
  month        = april,
  year         = 2020,
  publisher    = {Zenodo},
  version      = {v2.0.0},
  doi          = {10.5281/zenodo.3764169},
  url          = {https://doi.org/10.5281/zenodo.3764169}
}
```

DIMSpy v1.4.0

Ralf J. M. Weber & Jiarui Zhou. (2019, October 2). DIMSpy: Python package for processing direct-infusion mass spectrometry-based metabolomics and lipidomics data (Version v1.4.0). Zenodo. http://doi.org/10.5281/zenodo.3764110

BibTeX

```
@software{ralf_j_m_weber_2019_3764110,
  author       = {Ralf J. M. Weber and
                  Jiarui Zhou},
  title        = {{DIMSpy: Python package for processing direct-
                   infusion mass spectrometry-based metabolomics and
                   lipidomics data}},
  month        = oct,
  year         = 2019,
  publisher    = {Zenodo},
  version      = {v1.4.0},
  doi          = {10.5281/zenodo.3764110},
  url          = {https://doi.org/10.5281/zenodo.3764110}
}
```

1.8 License

Third-party licenses and copyright

CHAPTER

TWO

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

d

INDEX

F

G

H

I

J

L

M

O

P

R

S

T

U

V