## Contents

1 About spectrum_utils  
2 Citation  
3 Contents  
Python Module Index  
Index
Build status Python 3.6 Python 3.7 Python 3.8
spectrum_utils is a Python package for efficient MS/MS spectrum processing and visualization. spectrum_utils contains the following features:

• Spectrum processing
  – Precursor & noise peak removal
  – Intensity filtering
  – Intensity scaling
  – Peak annotations
    • Modification-aware (static & variable) peptide fragments
    • SMILES-based molecules
    • Custom strings

• Spectrum plotting
  – Fully customizable individual spectrum plots
  – Mirror plot of matching spectra
  – Interactive spectrum plots

See the documentation for more information and detailed examples on how to use this functionality.
spectrum_utils is freely available as open source under the Apache 2.0 license.

When using spectrum_utils please cite the following manuscript:

3.1 Install

`spectrum_utils`, including all its required dependencies, can be easily installed using conda from the Bioconda channel:

```
conda install -c conda-forge -c bioconda -c defaults spectrum_utils
```

3.1.1 Detailed installation information

**Supported Python versions**

`spectrum_utils` supports Python version 3.6 and above.

**Dependencies**

`spectrum_utils` has the following dependencies:

- Altair
- Matplotlib
- Numba
- NumPy
- Pandas
- Pyteomics
- RDKit

Missing dependencies will be automatically installed when you install `spectrum_utils` using conda.
3.1.2 Alternative installation options

The recommended way to install spectrum_utils is using conda. Alternatively, spectrum_utils can also be installed using pip:

```bash
pip install spectrum_utils
```

To install the basic spectrum_utils version. Or:

```bash
pip install spectrum_utils[plot]
```

To include the interactive plotting functionality (requires Pandas and Altair).

When installing using pip it is recommended to explicitly install any dependencies (listed above or in the environment file) in advance. Any missing dependencies will be automatically installed from PyPI when you install spectrum_utils, except RDKit. Please refer to the RDKit installation notes for information on how to install RDKit.

In contrast, when installing spectrum_utils using conda all dependencies will be automatically installed, including RDKit.

3.2 Quickstart

Here we briefly introduce spectrum_utils’ MS/MS spectrum processing and visualization functionality:

- Restrict the mass range to 100–1400 \( m/z \) to filter out irrelevant peaks.
- Remove the precursor peak.
- Remove low-intensity noise peaks by only retaining peaks that are at least 5\% of the base peak intensity and restrict the total number of peaks to the 50 most intense peaks.
- Scale the peak intensities by their square root to de-emphasize overly intense peaks.
- Annotate peaks corresponding to ‘b’, ‘y’, and ‘a’ peptide fragments.

IO functionality is not included in spectrum_utils. Instead you can use excellent libraries to read a variety of mass spectrometry data formats such as Pyteomics or pymzML.

```python
import matplotlib.pyplot as plt
import spectrum_utils.plot as sup
import spectrum_utils.spectrum as sus
from pyteomics import mgf

# Read the spectrum from an MGF file using Pyteomics.
spectrum_dict = mgf.get_spectrum(
    'spectra.mgf',
    'mzspec:PX004732:01650b_BC2-TUM_first_pool_53_01_01-3xHCD-1h-R2:scan:
     41840:WNQLQAFWGTGK/2')
identifier = spectrum_dict['params']['title']
precursor_mz = spectrum_dict['params']['pepmass'][0]
precursor_charge = spectrum_dict['params']['charge'][0]
mz = spectrum_dict['m/z array']
intensity = spectrum_dict['intensity array']
retention_time = float(spectrum_dict['params']['rtinseconds'])
peptide = 'WNQLQAFWGTGK'

# Create the MS/MS spectrum.
```

(continues on next page)
As demonstrated, each of the processing steps can be achieved using a single, high-level function call. These calls can be chained together to easily perform multiple processing steps.

Note that several processing steps modify the peak \( m/z \) and intensity values and are thus not idempotent.

Spectrum plotting can similarly be achieved using a high-level function call, resulting in the following figure:

3.3 Spectrum processing

See the quickstart for a brief introduction to how to start using spectrum_utils. Here we will describe the spectrum processing functionality provided by spectrum_utils in more detail.
3.3.1 Peak annotations

Fragment ions can be annotated as follows:

- Using `MsmsSpectrum.annotate_peptide_fragments(...)` to annotate a, b, c, x, y, or z ions for peptide spectra.
- Using `MsmsSpectrum.annotate_molecule_fragment(...)` using SMILES strings to annotate peaks with molecule (sub)structures.
- Using `MsmsSpectrum.annotate_mz_fragment(...)` to annotate peaks with their \( m/z \) value or user-provided custom strings.

Peak annotations can be visualized using the `spectrum_utils` plotting functionality.

The example in the `quickstart` shows how spectrum peaks can be annotated with peptide fragments.

The following example shows how spectrum peaks can be annotated with their \( m/z \) values and a SMILES-based substructure:

```
import matplotlib.pyplot as plt
import spectrum_utils.spectrum as sus
import spectrum_utils.plot as sup
from pyteomics import mgf

spectrum_dict = mgf.get_spectrum('spectra.mgf', 'CCMSLIB00000840351')
identifier = spectrum_dict['params']['title']
precursor_mz = spectrum_dict['params']['pepmass'][0]
precursor_charge = spectrum_dict['params']['charge'][0]
mz = spectrum_dict['m/z array']
intensity = spectrum_dict['intensity array']

spectrum = sus.MsmsSpectrum(
    identifier, precursor_mz, precursor_charge, mz, intensity)
spectrum.filter_intensity(0.05)

charge, tol_mass, tol_mode = 1, 0.5, 'Da'
annotate_fragment_mz = [133.102, 147.080, 195.117, 237.164, 267.174, 295.170,
                        313.181, 355.192, 377.172, 391.187, 451.209, 511.231,
                        573.245, 633.269]

for fragment_mz in annotate_fragment_mz:
    spectrum.annotate_mz_fragment(fragment_mz, charge, tol_mass, tol_mode)

fragment_smiles = '\\\n'.join(['[H][C@\(8\)]1([C@\(8\)]([C@\(8\)]([H])([C@\(8\)]([H])\(C4(C)C\)'\
                        'C=C[C@@\(8\)]13C=C5C2=C[C@\(+]\(C5\)C']

fragment_mz = 295.170
spectrum.annotate_molecule_fragment(fragment_smiles, fragment_mz, charge,
                                      tol_mass, tol_mode)

fig, ax = plt.subplots(figsize=(12, 6))
sup.spectrum(spectrum, ax=ax)
plt.show()
plt.close()
```

Resulting in the following spectrum plot:
3.3.2 Peptide modifications

When annotating peptide spectra the masses of the fragment ions will be automatically calculated to annotate the corresponding peaks. This functionality is modification-aware: you can specify static modifications to globally change the mass of specific amino acid residues and variable modifications to modify amino acids in specific positions for a given peptide.

**Static modifications**

Static modifications can be set as follows:

```python
sus.static_modification('C', 57.02146)
```

To, for example, set a static carbamidomethylation of cysteine. Modification mass differences can be either positive or negative.

All static modifications can be reset:

```python
sus.reset_modifications()
```

**Variable modifications**

Variable modifications can be set for an individual spectrum and peptide by specifying the amino acid index and corresponding mass difference for each modification.

Modification positions can be the following:

- The index of the amino acid where the modification is present (0-based).
- ‘N-term’ for N-terminal modifications.
- ‘C-term’ for C-terminal modifications.
3.4 Spectrum visualization

The quickstart and spectrum processing pages include several examples that demonstrate the spectrum_utils plotting functionality. Often, nice spectrum graphics can be created with a single line of code; it’s as easy as using spectrum_utils.plot.spectrum(...) to visualize a single spectrum. Additionally, these examples show how peaks will be annotated with peptide fragments, molecule structures, their m/z values, or custom strings.

Here we will briefly describe some functionality to customize your spectrum plots. Some of the arguments that can be provided to spectrum_utils.plot.spectrum(...) are:

- **color_ions** and **annotate_ions**: Boolean flags indicating whether the peaks should be colored and/or annotated.
- **annot_kws**: A dictionary with options to customize textual peak annotations. See the matplotlib.text documentation for available options.
- **grid**: Enable/disable the grid.

See the API reference for more information.

### 3.4.1 Mirror plot

A mirror plot can be used to visualize matching spectra, for example, to plot identifications from spectral library searching. Again, only a single line of code is required to do the actual plotting: spectrum_utils.plot.mirror(...)
precursor_charge, mz, intensity,
retention_time=retention_time,
peptide=peptide,
modifications=modifications)
    .filter_intensity(0.01, 50)
    .scale_intensity('root')
    .annotate_peptide_fragments(0.5, 'Da', ion_types='aby'))

fig, ax = plt.subplots(figsize=(12, 6))
spectrum_top, spectrum_bottom = spectra
sup.mirror(spectrum_top, spectrum_bottom, ax=ax)
plt.show()
plt.close()

All of the advanced plotting arguments described above can be provided for the mirror plot as well using the `spectrum_kws` argument.

### 3.4.2 Interactive plotting

Besides the standard plotting functionality in `spectrum_utils.plot`, `spectrum_utils` also contains dynamic plotting functionality in `spectrum_utils.iplot`. `iplot` is a drop-in replacement for `plot`; only the import statement needs to be changed to generate interactive plots.

See below for example interactive plots of a single spectrum and a mirror plot.

```python
import spectrum_utils.iplot as sup
import spectrum_utils.spectrum as sus
from pyteomics import mgf

# Read the spectrum from an MGF file using Pyteomics.
spectrum_dict = mgf.get_spectrum('spectra.mgf',
    'spectra.mgf',

(continues on next page)
identifier = spectrum_dict['params']['title']
precursor_mz = spectrum_dict['params']['pepmass'][0]
precursor_charge = spectrum_dict['params']['charge'][0]
mz = spectrum_dict['m/z array']
intensity = spectrum_dict['intensity array']
retention_time = float(spectrum_dict['params']['rtinseconds'])
peptide = 'WNQLQAFWGTGK'

# Create the MS/MS spectrum.
spectrum = sus.MsmsSpectrum(identifier, precursor_mz, precursor_charge, mz, intensity,
retention_time=retention_time, peptide=peptide)

# Process the MS/MS spectrum.
fragment_tol_mass = 10
fragment_tol_mode = 'ppm'
spectrum = (spectrum.set_mz_range(min_mz=100, max_mz=1400)
.remove_precursor_peak(fragment_tol_mass, fragment_tol_mode)
.filter_intensity(min_intensity=0.05, max_num_peaks=50)
.scale_intensity('root')
.annotate_peptide_fragments(fragment_tol_mass, fragment_tol_mode,
ion_types='aby'))

# Plot the MS/MS spectrum.
(sup.spectrum(spectrum).properties(width=640, height=400)
.save('spectrum_iplot.json'))
For more information on how to manipulate these interactive plots, see the Altair documentation.

Interactive plots can be saved as html files or other output formats. Vega-Embed makes it easy to embed plots exported as JSON into web pages.

### 3.4.3 Advanced functionality

#### Peak colors

By default, peaks are colored as follows:

- a-ions
- b-ions
- c-ions
- x-ions
- y-ions
- z-ions
- unknown and unannotated ions

To change these colors, simply overwrite values in the `spectrum_utils.plot.colors` dictionary with your preferred colors:

```python
import spectrum_utils.plot as sup
sup.colors['y'] = '#FF1493'
```

### 3.5 Computational efficiency

Spectrum processing in `spectrum_utils` has been optimized for computational efficiency using NumPy and Numba to be able to process thousands of spectra per second.

As shown below, `spectrum_utils` is faster than alternative libraries, such as pymzML (version 2.4.4) and pyOpenMS (version 2.4.0), when performing typical spectrum processing tasks, including the following steps:

- The $m/z$ range is set to 100–1400 $m/z$.
- The precursor peak is removed.
- Low-intensity noise peaks are removed.
- Peak intensities are scaled by their square root.

```python
import time
import matplotlib.pyplot as plt
import numpy as np
```

(continues on next page)
import pyopenms
import seaborn as sns
import spectrum_utils.spectrum as sus
from pymzml.spec import Spectrum
from pyteomics import mgf

min_peaks = 10
min_mz, max_mz = 100, 1400
fragment_tol_mass, fragment_tol_mode = 0.02, 'Da'
min_intensity = 0.05
max_num_peaks = 150

def time_spectrum_utils(mgf_filename):
    runtimes = []
    for mgf_in in mgf.read(mgf_filename):
        # Omit invalid spectra.
        if (len(mgf_in['m/z array']) < min_peaks or
            'charge' not in mgf_in['params']):
            continue
        mz = mgf_in['m/z array']
        intensity = mgf_in['intensity array']
        retention_time = float(mgf_in['params']['rtinseconds'])
        precursor_mz = mgf_in['params']['pepmass'][0]
        precursor_charge = mgf_in['params']['charge'][0]
        identifier = mgf_in['params']['title']

        spectrum = sus.MsmsSpectrum(
            identifier, precursor_mz, precursor_charge, mz, intensity,
            retention_time=retention_time)

        start_time = time.time()

        (spectrum.set_mz_range(min_mz, max_mz)
         .remove_precursor_peak(fragment_tol_mass, fragment_tol_mode)
         .filter_intensity(min_intensity, max_num_peaks)
         .scale_intensity(scaling='root', max_intensity=1))

        runtimes.append(time.time() - start_time)

    return runtimes

def time_pymzml(mgf_filename):
    runtimes = []
    for mgf_in in mgf.read(mgf_filename):
        # Omit invalid spectra.
        if (len(mgf_in['m/z array']) < min_peaks or
            'charge' not in mgf_in['params']):
            continue

        spec = Spectrum()
        spec.set_peaks(
            [*zip(mgf_in['m/z array'], mgf_in['intensity array'])], 'raw')

        start_time = time.time()
```
spec.reduce('raw', (min_mz, max_mz))
spec.remove_precursor_peak()
spec.remove_noise(noise_level=min_intensity)
spec /= np.amax(spec.i)
spec.i = np.sqrt(spec.i)

runtimes.append(time.time() - start_time)

return runtimes

def time_pyopenms(mgf_filename):
    experiment = pyopenms.MSExperiment()
    pyopenms.MascotGenericFile().load(mgf_filename, experiment)

    runtimes = []
    for spectrum in experiment:
        # Omit invalid spectra.
        if (len(spectrum.get_peaks())[0]) < min_peaks or  
            spectrum.getPrecursors()[0].getCharge() == 0):
            continue

        start_time = time.time()
        # Set the m/z range.
        filtered_mz, filtered_intensity = [], []
        for mz, intensity in spectrum.get_peaks():
            if min_mz <= mz <= max_mz:
                filtered_mz.append(mz)
                filtered_intensity.append(intensity)
        spectrum.set_peaks((filtered_mz, filtered_intensity))

        # Remove the precursor peak.
        parent_peak_mower = pyopenms.ParentPeakMower()
        parent_peak_mower_params = parent_peak_mower.getDefaults()
        parent_peak_mower_params.setValue(b'window_size', fragment_tol_mass, b'')
        parent_peak_mower.setParameters(parent_peak_mower_params)
        parent_peak_mower.filterSpectrum(spectrum)

        # Filter by base peak intensity percentage.
        pyopenms.Normalizer().filterSpectrum(spectrum)
        threshold_mower = pyopenms.ThresholdMower()
        threshold_mower_params = threshold_mower.getDefaults()
        threshold_mower_params.setValue(b'threshold', min_intensity, b'')
        threshold_mower.setParameters(threshold_mower_params)
        threshold_mower.filterSpectrum(spectrum)

        # Restrict to the most intense peaks.
        n_largest = pyopenms.NLargest()
        n_largest_params = n_largest.getDefaults()
        n_largest_params.setValue(b'n', max_num_peaks, b'')
        n_largest.setParameters(n_largest_params)
        n_largest.filterSpectrum(spectrum)

        # Scale the peak intensities by their square root and normalize.
        pyopenms.SqrtMower().filterSpectrum(spectrum)
        pyopenms.Normalizer().filterSpectrum(spectrum)

        runtimes.append(time.time() - start_time)
```

3.5. Computational efficiency 17
return runtimes

gf_filename = 'iPRG2012.mgf'
runtimes_spectrum_utils = time_spectrum_utils(mgf_filename)
runtimes_pymzml = time_pymzml(mgf_filename)

fig, ax = plt.subplots()
sns.boxplot(data=[runtimes_spectrum_utils, runtimes_pymzml,
    runtimes_pyopenms], flierprops={'markersize': 2}, ax=ax)
ax.set_yscale('log')
ax.xaxis.set_ticklabels(('spectrum_utils', 'pymzML', 'pyOpenMS'))
ax.set_ylabel('Processing time per spectrum (s)')
sns.despine()
plt.show()
plt.close()

3.5.1 JIT compilation

Note that the significant outlier for spectrum_utils is caused by Numba’s JIT compilation of the first method call, allowing subsequent calls to be made very efficiently.

If the user knows in advance that only a single method call needs to be made, Numba’s JIT compilation can be disabled to avoid this overhead by setting the NUMBA_DISABLE_JIT environment variable to 1. See the Numba documentation for more information.
# 3.6 spectrum_utils package

## 3.6.1 spectrum_utils.spectrum module

class spectrum_utils.spectrum.FragmentAnnotation(charge: int, calc_mz: float, annotation: str = None)

Bases: object

Class representing a general fragment ion annotation.

__init__(charge: int, calc_mz: float, annotation: str = None) → None

Instantiate a new FragmentAnnotation.

Parameters

• **charge** (int) – The fragment ion charge if known, None otherwise.

• **calc_mz** (float) – The theoretical m/z value of the fragment.

• **annotation** (str) – The fragment’s annotation string.

class spectrum_utils.spectrum.MoleculeFragmentAnnotation(charge: int, calc_mz: float, smiles: str)

Bases: spectrum_utils.spectrum.FragmentAnnotation

Class representing a molecule fragment ion annotation.

__init__(charge: int, calc_mz: float, smiles: str) → None

Instantiate a new MoleculeFragmentAnnotation.

Parameters

• **charge** (int) – The molecule fragment ion charge.

• **calc_mz** (float) – The theoretical m/z value of the molecule fragment.

• **smiles** (str) – The SMILES representation of the molecule.


Bases: object

Class representing a tandem mass spectrum.

__init__(identifier: str, precursor_mz: float, precursor_charge: int, mz: Union[numpy.ndarray, Iterable[T_co]], intensity: Union[numpy.ndarray, Iterable[T_co]], annotation: Union[numpy.ndarray, Iterable[T_co], None] = None, retention_time: Optional[float] = None, peptide: Optional[str] = None, modifications: Optional[Dict[Union[int, str], float]] = None, is_decoy: bool = False) → None

Instantiate a new MsmsSpectrum consisting of fragment peaks.

Parameters

• **identifier** (str) – (Unique) spectrum identifier. See for example the universal spectrum identifier specification by the Proteomics Standards Initiative.
• **precursor_mz** (*float*) – Precursor ion mass-to-charge ratio.

• **precursor_charge** (*int*) – Precursor ion charge.

• **mz** (*array_like*) – Mass-to-charge ratios of the fragment peaks.

• **intensity** (*array_like*) – Intensities of the corresponding fragment peaks in *mz*.

• **annotation** (*Optional*[*,*array_like*,*], *optional*) – Annotations of the corresponding fragment peaks in *mz* (the default is None, which indicates that the fragment peaks are not annotated).

• **retention_time** (*Optional*[*,*float*,*], *optional*) – Retention time at which the spectrum was acquired (the default is None, which indicates that retention time is unspecified/unknown).

• **peptide** (*Optional*[*,*str*,*], *optional*) – The peptide sequence corresponding to the spectrum (the default is None, which means that no peptide-spectrum match is specified). The peptide sequence should only exist of the 20 standard amino acids.

• **modifications** (*Optional*[*,*Dict*[*,*Union*[*,*int*,*str*],*float*],*], *optional*) – Mapping of modification positions and mass differences. Valid positions are any amino acid index in the peptide (0-based), ‘N-term’, and ‘C-term’. 

• **is_decoy** (*bool*, *optional*) – Flag indicating whether the peptide is a target or decoy peptide (the default is False, which implies a target peptide).

### annotate_molecule_fragment

*smiles*: *str*, *fragment_mz*: *float*, *fragment_charge*: *int*, *fragment_tol_mass*: *float*, *fragment_tol_mode*: *str*, *peak_assignment*: *str* = *most_intense*) → spectrum_utils.spectrum.MsmsSpectrum

Annotate a peak (if present) with its corresponding molecule.

The matching position in *self*.*annotation* will be overwritten by the provided molecule.

**Parameters**

• **smiles** (*str*) – The fragment molecule that will be annotated in SMILES format.

• **fragment_mz** (*float*) – The expected m/z of the molecule.

• **fragment_charge** (*int*) – The charge of the molecule.

• **fragment_tol_mass** (*float*) – Fragment mass tolerance to match spectrum peaks against the theoretical molecule mass.

• **fragment_tol_mode** (*{'Da', 'ppm'}*) – Fragment mass tolerance unit. Either ‘Da’ or ‘ppm’.

• **peak_assignment** (*{'most_intense', 'nearest_mz'}, *optional*) – In case multiple peaks occur within the given mass window around the molecule’s given mass, only a single peak will be annotated:
  – ‘most_intense’: The most intense peak will be annotated (default).
  – ‘nearest_mz’: The peak whose m/z is closest to the theoretical m/z will be annotated.

**Returns** self

**Return type** MsmsSpectrum

### annotate_mz_fragment

*fragment_mz*: *float*, *fragment_charge*: *int*, *fragment_tol_mass*: *float*, *fragment_tol_mode*: *str*, *peak_assignment*: *str* = *most_intense*, *text*: *Optional*[*,*str*,*] = *None*) → spectrum_utils.spectrum.MsmsSpectrum

Annotate a peak (if present) with its m/z value or a custom provided string.
The matching position in `self.annotation` will be overwritten.

**Parameters**

- `fragment_mz (float)` – The expected m/z to annotate.
- `fragment_charge (int)` – The peak charge.
- `fragment_tol_mass (float)` – Fragment mass tolerance to match spectrum peaks against the given m/z.
- `fragment_tol_mode ({'Da', 'ppm'})` – Fragment mass tolerance unit. Either 'Da' or 'ppm'.
- `peak_assignment ({'most_intense', 'nearest_mz'}, optional)` – In case multiple peaks occur within the given mass window around the given m/z, only a single peak will be annotated:
  - 'most_intense': The most intense peak will be annotated (default).
  - 'nearest_mz': The peak whose m/z is closest to the given m/z will be annotated.
- `text (Optional[str], optional)` – The text to annotate the peak with. If None, its m/z value will be used.

**Returns** self

**Return type** MsmsSpectrum

```python
annotate_peaks(*args, **kwargs)
```

```python
annotate_peptide_fragments(fragment_tol_mass: float, fragment_tol_mode: str, ion_types: str = 'by', max_ion_charge: Optional[int] = None, peak_assignment: str = 'most_intense') → spectrum_utils.spectrum.MsmsSpectrum
```

Annotate peaks with their corresponding peptide fragment ion annotations.

`self.annotation` will be overwritten and include `PeptideFragmentAnnotation` objects for matching peaks.

**Parameters**

- `fragment_tol_mass (float)` – Fragment mass tolerance to match spectrum peaks against theoretical peaks.
- `fragment_tol_mode ({'Da', 'ppm'})` – Fragment mass tolerance unit. Either 'Da' or 'ppm'.
- `ion_types (str, optional)` – Fragment type to annotate. Can be any combination of 'a', 'b', 'c', 'x', 'y', and 'z' (the default is 'by', which means that b-ions and y-ions will be annotated).
- `max_ion_charge (Optional[int], optional)` – All fragments up to and including the given charge will be annotated (by default all fragments with a charge up to the precursor minus, or minimum charge one, one will be annotated).
- `peak_assignment ({'most_intense', 'nearest_mz'}, optional)` – In case multiple peaks occur within the given mass window around a theoretical peak, only a single peak will be annotated with the fragment type:
  - 'most_intense': The most intense peak will be annotated (default).
  - 'nearest_mz': The peak whose m/z is closest to the theoretical m/z will be annotated.

**Returns** self

**Return type** MsmsSpectrum
**annotation**
Get or set the annotations of the fragment peaks.

When setting new annotations it should be possible to convert the given values to a NumPy array (or None) and the number of annotations should be equal to the number of m/z and intensity values.

**Returns** The annotations of the fragment peaks or None if no annotations have been specified.

**Return type** Optional[np.ndarray]

**filter_intensity** *(min_intensity: float = 0.0, max_num_peaks: Optional[int] = None) → spectrum_utils.spectrum.MsmsSpectrum*
Remove low-intensity fragment peaks.

Only the `max_num_peaks` most intense fragment peaks are retained. Additionally, noise peaks whose intensity is below `min_intensity` percentage of the intensity of the most intense peak are removed.

**Parameters**

- **min_intensity** *(float, optional)* – Remove peaks whose intensity is below `min_intensity` percentage of the intensity of the most intense peak (the default is 0, which means that no minimum intensity filter will be applied).

- **max_num_peaks** *(Optional[int], optional)* – Only retain the `max_num_peaks` most intense peaks (the default is None, which retains all peaks).

**Returns** self

**Return type** MsmsSpectrum

**intensity**
Get or set the intensities of the fragment peaks.

When setting new intensity values it should be possible to convert the given values to a NumPy array and the number of intensity values should be equal to the number of m/z (and annotation) values.

**Returns** The intensity values of the fragment peaks.

**Return type** np.ndarray

**mz**
Get or set the mass-to-charge ratios of the fragment peaks.

When setting new m/z values it should be possible to convert the given values to a NumPy array and the number of m/z values should be equal to the number of intensity (and annotation) values.

**Returns** The mass-to-charge ratios of the fragment peaks.

**Return type** np.ndarray

**remove_precursor_peak** *(fragment_tol_mass: float, fragment_tol_mode: str, isotope: int = 0) → spectrum_utils.spectrum.MsmsSpectrum*
Remove fragment peak(s) close to the precursor mass-to-charge ratio.

**Parameters**

- **fragment_tol_mass** *(float)* – Fragment mass tolerance around the precursor mass to remove the precursor peak.

- **fragment_tol_mode** *(str)* – Fragment mass tolerance unit. Either ‘Da’ or ’ppm’.

- **isotope** *(int)* – The number of precursor isotopic peaks to be checked (the default is 0 to check only the mono-isotopic peaks).

**Returns** self
Return type **MsmsSpectrum**

**round**(*decimals: int = 0, combine: str = 'sum') → spectrum_utils.spectrum.MsmsSpectrum

Round the mass-to-charge ratios of the fragment peaks to the given number of decimals. Peaks that have the same mass-to-charge ratio after rounding will be combined using the specified strategy. If multiple peaks are merged into a single peak it will be annotated with the annotation of the most intense peak.

**Parameters**

- **decimals** (*int, optional*) – Number of decimal places to round the *mz* to (default: 0). If decimals is negative, it specifies the number of positions to the left of the decimal point.
- **combine** (*{'sum', 'max'}*) – Method used to combine intensities from merged fragment peaks. *sum* specifies that the intensities of the merged fragment peaks will be summed, *max* specifies that the maximum intensity of the fragment peaks that are merged is used (the default is *sum*).

**Returns** self

Return type **MsmsSpectrum**

**scale_intensity**(*scaling: Optional[str] = None, max_intensity: Optional[float] = None, **kwargs) → spectrum_utils.spectrum.MsmsSpectrum

Scale the intensity of the fragment peaks. Two types of scaling can be performed: scaling all peaks using a specific transformation and scaling the peaks relative to the most intense peak.

**Parameters**

- **scaling** (*{'root', 'log', 'rank'}, optional*) – Method to scale the peak intensities (the default is None, which means that no transformation will be performed). Potential transformation options are:
  - *root*: Root-transform the peak intensities. The default is a square root transformation (*degree* is 2). The degree of the root can be specified using the *degree* kwarg.
  - *log*: Log-transform the peak intensities. The default is a log2 transformation (*base* is 2) after summing the intensities with 1 to avoid negative values after the transformation. The base of the logarithm can be specified using the *base* kwarg.
  - *rank*: Rank-transform the peak intensities. The maximum rank of the most intense peak can be specified using the *max_rank* kwarg, by default the number of peaks in the spectrum is used as the maximum rank. Note that *max_rank* should be greater than or equal to the number of peaks in the spectrum.
- **max_intensity** (*Optional[float], optional*) – Intensity of the most intense peak relative to which the peaks will be scaled (the default is None, which means that no scaling relative to the most intense peak will be performed).

**Returns** self

Return type **MsmsSpectrum**

**set_mz_range**(*min_mz: Optional[float] = None, max_mz: Optional[float] = None) → spectrum_utils.spectrum.MsmsSpectrum

Restrict the mass-to-charge ratios of the fragment peaks to the given range.

**Parameters**
• **min_mz** *(Optional[float], optional)* – Minimum m/z (inclusive). If not set no minimal m/z restriction will occur.

• **max_mz** *(Optional[float], optional)* – Maximum m/z (inclusive). If not set no maximal m/z restriction will occur.

Returns: self

Return type: MsmsSpectrum

class spectrum_utils.spectrum.PeptideFragmentAnnotation

Bases: spectrum_utils.spectrum.FragmentAnnotation

Class representing a peptide fragment ion annotation.

__init__ *(charge: int, calc_mz: float, ion_type: str, ion_index: int) → None*

Instantiate a new PeptideFragmentAnnotation.

Parameters

• **charge** *(int)* – The peptide fragment ion charge.

• **calc_mz** *(float)* – The theoretical m/z value of the peptide fragment.

• **ion_type** *(\{'a', 'b', 'c', 'x', 'y', 'z'\})* – The peptide fragment ion type.

• **ion_index** *(int)* – The peptide fragment ion index.

spectrum_utils.spectrum.reset_modifications() → None

Undo all static modifications and reset to the standard amino acid monoisotopic masses.

spectrum_utils.spectrum.static_modification *(amino_acid: str, mass_diff: float) → None*

Globally modify the monoisotopic mass of an amino acid to set a static modification.

Parameters

• **amino_acid** *(str)* – The amino acid whose monoisotopic mass is modified.

• **mass_diff** *(float)* – The mass difference to be added to the amino acid’s original monoisotopic mass.

3.6.2 spectrum_utils.plot module


Mirror plot two MS/MS spectra.

Parameters

• **spec_top** *(MsmsSpectrum)* – The spectrum to be plotted on the top.

• **spec_bottom** *(MsmsSpectrum)* – The spectrum to be plotted on the bottom.

• **spectrum_kws** *(Optional[Dict], optional)* – Keyword arguments for plot.spectrum.

• **ax** *(Optional[plt.Axes], optional)* – Axes instance on which to plot the spectrum. If None the current Axes instance is used.
Returns
The matplotlib Axes instance on which the spectra are plotted.

Return type
plt.Axes


Plot an MS/MS spectrum.

Parameters
- **spec** (MsmsSpectrum) – The spectrum to be plotted.
- **color_ions** (bool, optional) – Flag indicating whether or not to color annotated fragment ions. The default is True.
- **annotate_ions** (bool, optional) – Flag indicating whether or not to annotate fragment ions. The default is True.
- **annot_kws** (Optional[Dict], optional) – Keyword arguments for ax.text to customize peak annotations.
- **mirror_intensity** (bool, optional) – Flag indicating whether to flip the intensity axis or not.
- **grid** (Union[bool, str], optional) – Draw grid lines or not. Either a boolean to enable/disable both major and minor grid lines or ‘major’/’minor’ to enable major or minor grid lines respectively.
- **ax** (Optional[plt.Axes], optional) – Axes instance on which to plot the spectrum. If None the current Axes instance is used.

Returns
The matplotlib Axes instance on which the spectrum is plotted.

Return type
plt.Axes

3.6.3 spectrum_utils.iplot module

spectrum_utils.iplot_mirror(spec_top: spectrum_utils.spectrum.MsmsSpectrum, spec_bottom: spectrum_utils.spectrum.MsmsSpectrum, spectrum_kws: Optional[Dict[KT, VT]] = None, *_) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f39f328b080>

Mirror plot two MS/MS spectra.

Parameters
- **spec_top** (MsmsSpectrum) – The spectrum to be plotted on the top.
- **spec_bottom** (MsmsSpectrum) – The spectrum to be plotted on the bottom.
- **spectrum_kws** (Optional[Dict], optional) – Keyword arguments for iplot.spectrum.
- **_** – Ignored, for consistency with the plot_mirror API.

Returns
The Altair chart instance with the plotted spectrum.

Return type
altair.LayerChart

3.6. spectrum_utils package
Plot an MS/MS spectrum.

**Parameters**

- **spec** (*MsmsSpectrum*) – The spectrum to be plotted.
- **color_ions** (*bool, optional*) – Flag indicating whether or not to color annotated fragment ions. The default is True.
- **annotate_ions** (*bool, optional*) – Flag indicating whether or not to annotate fragment ions. The default is True.
- **annot_kws** (*Optional[Dict], optional*) – Keyword arguments for *altair.Chart.mark_text* to customize peak annotations.
- **mirror_intensity** (*bool, optional*) – Flag indicating whether to flip the intensity axis or not.
- **grid** (*bool, optional*) – Draw grid lines or not.
- **_** – Ignored, for consistency with the *plot.spectrum* API.

**Returns** The *Altair* chart instance with the plotted spectrum.

**Return type** *altair.LayerChart*

### 3.6.4 spectrum_utils.utils module

#### spectrum_utils.utils.mass_diff

Calculate the mass difference(s).

**Parameters**

- **mz1** – First m/z value(s).
- **mz2** – Second m/z value(s).
- **mode_is_da** (*bool*) – Mass difference in Dalton (True) or in ppm (False).

**Returns**

**Return type** The mass difference(s) between the given m/z values.

### 3.7 Contact

For more information you can visit the [official GitHub repository](https://github.com/).

### 3.7.1 Citation

When using *spectrum_utils* please cite the following manuscript:

S
spectrum_utils.iplot, 25
spectrum_utils.plot, 24
spectrum_utils.spectrum, 19
spectrum_utils.utils, 26
Symbols

__init__() (spectrum_utils.spectrum.FragmentAnnotation method), 19
__init__() (spectrum_utils.spectrum.MoleculeFragmentAnnotation method), 19
__init__() (spectrum_utils.spectrum.MsmsSpectrum method), 19
__init__() (spectrum_utils.spectrum.PeptideFragmentAnnotation method), 24
annotate_molecule_fragment() (spectrum_utils.spectrum.MsmsSpectrum method), 20
annotate_mz_fragment() (spectrum_utils.spectrum.MsmsSpectrum method), 20
annotate_peaks() (spectrum_utils.spectrum.MsmsSpectrum method), 21
annotate_peptide_fragments() (spectrum_utils.spectrum.MsmsSpectrum method), 21
annotation (spectrum_utils.spectrum.MsmsSpectrum attribute), 21

F
filter_intensity() (spectrum_utils.spectrum.MsmsSpectrum method), 22
FragmentAnnotation (class in spectrum_utils.spectrum), 19

I
intensity (spectrum_utils.spectrum.MsmsSpectrum attribute), 22

M
mass_diff (in module spectrum_utils.utils), 26

mirror() (in module spectrum_utils.iplot), 25
mirror() (in module spectrum_utils.plot), 24
MoleculeFragmentAnnotation (class in spectrum_utils.spectrum), 19
MsmsSpectrum (class in spectrum_utils.spectrum), 19
mz (spectrum_utils.spectrum.MsmsSpectrum attribute), 22
PeptideFragmentAnnotation (class in spectrum_utils.spectrum), 24
remove_precursor_peak() (spectrum_utils.spectrum.MsmsSpectrum method), 22
reset_modifications() (in module spectrum_utils.spectrum), 22
round() (spectrum_utils.spectrum.MsmsSpectrum method), 24

S
scale_intensity() (spectrum_utils.spectrum.MsmsSpectrum method), 23
set_mz_range() (spectrum_utils.spectrum.MsmsSpectrum method), 23
spectr(...)