pyQms Documentation

Release 0.5.0-beta

Johannes Leufken, Anna Niehues, L. Peter Sarin, Michael Hippler,

Jun 03, 2019

Contents

The latest Documentation was generated on: Jun 03, 2019

CHAPTER 1

Introduction

pyQms enables universal and accurate quantification of mass spectrometry data

1.1 Summary

pyQms is an extension to Python that offers amongst other things

a) fast and accurate quantification of all high-res LC-MS data

b) full labeling and modification flexibility

c) full platform independence

1.2 Abstract

Quantitative mass spectrometry (MS) is a key technique in many research areas (Yates III et al. 2009), including proteomics, metabolomics, glycomics, and lipidomics. Because all of the corresponding molecules can be described by chemical formulas, universal quantification tools are highly desirable. Here we present pyQms, an open-source software for accurate quantification of all types of molecules measurable by MS. pyQms uses isotope pattern matching which offers accurate quality assessment of the quantification and the ability to directly incorporate mass spectrometer accuracy. pyQms is, due to its universal design, applicable to every research field, labeling strategy, and acquisition technique. This opens ultimate flexibility for researchers to design experiments employing innovative and hitherto unexplored labeling strategies. Importantly, pyQms performs very well to accurately quantify partially labeled proteomes in large-scale and high-throughput, the most challenging task for a quantification algorithm.

> – Leufken, J., Niehues, A., Hippler, M., Sarin, L. P., Hippler, M., Leidel, S. A., and Fufezan, C. (2017) pyQms enables universal and accurate quantification of mass spectrometry data. MCP In Press

Link to manuscript.

http://www.mcponline.org/content/early/2017/07/20/mcp.M117.068007.abstract

1.3 pyQms module

At its core, pyQms is a Python module that allows a isotope pattern library to be initialized and any list of (mz, intensity) to be matched against the library, yielding a mScore.

1.4 Documentation

http://pyqms.readthedocs.io/en/latest/

1.5 Implementation

pyQms requires Python3.4+ .

The module is freely available on pyqms.github.io or pypi, published under MIT LGPL and requires no additional modules to be installed. For fast spectra from mzML access we recommend pymzML (Bald et al. 2012). For example scripts it is necessary to install pymzML as well or change the code for alternated spectra access. For some scripts also the openpyxl module is required.

1.6 Download

Get the latest version via github

https://github.com/pyQms/pyQms

1.7 Citation

Please cite us when using pyQms in your work.

The original publication can be found here: Leufken, J., Niehues, A., Hippler, M., Sarin, L. P., Hippler, M., Leidel, S. A., and Fufezan, C. (2017) pyQms enables universal and accurate quantification of mass spectrometry data. Mol. Cell. Proteomics 16, 1736–1745

1.7.1 Full article

http://www.mcponline.org/content/16/10/1736

1.7.2 Early access article version

http://www.mcponline.org/content/early/2017/07/20/mcp.M117.068007.abstract

1.7.3 DOI

10.1074/mcp.M117.068007

1.8 Installation

Install requirements:

```
user@localhost:~$ cd pyqms
user@localhost:~/pyqms$ pip3.4 install -r requirements.txt
```

Install pyQms:

```
user@localhost:~/pyqms$ python3.4 setup.py install
```

pyQms can be also be installed via pip:

```
pip install pyqms
```

(You might need administrator privileges to write in the Python site-package folder. On Linux or OS X, use `sudo python setup.py install` or write into a user folder by using this command `python setup.py install --user`. On Windows, you have to start the command line with administrator privileges.)

1.8.1 pyQms docs recompiling and extending

You will require sphinx and other packages to build the documentation from scratch. We recommend to use a Python virtual environment for easy installation and use.

1.9 Tests

Run nosetests in root folder. You might need to install nose for Python3 first. Then just execute:

```
user@localhost:~/pyqms$ nosetests3
```

to test the package.

1.10 LICENSE

This software is under MIT license, please refer to LICENSE for full license.

1.11 Publications and project using pyQms for quantification

- Hohner, R., Barth, J., Magneschi, L., Jaeger, D., Niehues, A., Bald, T., Grossman, A., Fufezan, C., and Hippler, M. (2013) The Metabolic Status Drives Acclimation of Iron Deficiency Responses in Chlamydomonas reinhardtii as Revealed by Proteomics Based Hierarchical Clustering and Reverse Genetics. **Mol. Cell. Proteomics** 12, 2774–2790 Pubmed

- Barth, J., Bergner, S. V., Jaeger, D., Niehues, A., Schulze, S., Scholz, M., and Fufezan, C. (2014) The Interplay of Light and Oxygen in the Reactive Oxygen Stress Response of Chlamydomonas reinhardtii Dissected by Quantitative Mass Spectrometry. **Mol. Cell. Proteomics** 13, 969–989 Pubmed

- Kukuczka, B., Magneschi, L., Petroutsos, D., Steinbeck, J., Bald, T., Powikrowska, M., Fufezan, C., Finazzi, G., and Hippler, M. (2014) Proton Gradient Regulation5-Like1-Mediated Cyclic Electron Flow Is Crucial for Acclimation to Anoxia and Complementary to Nonphotochemical Quenching in Stress Adaptation. **Plant Physiol.** 165, 1604–1617 Pubmed

- Alings, F., Sarin, L. P., Fufezan, C., Drexler, H. C. A., and Leidel, S. A. (2015) An evolutionary approach uncovers a diverse response of tRNA 2-thiolation to elevated temperatures in yeast. **RNA** 21, 202–212 Pubmed

- Bergner, S. V., Scholz, M., Trompelt, K., Barth, J., Gäbelein, P., Steinbeck, J., Xue, H., Clowez, S., Fucile, G., Goldschmidt-Clermont, M., Fufezan, C., and Hippler, M. (2015) STATE TRANSITION7-Dependent Phosphorylation Is Modulated by Changing Environmental Conditions, and Its Absence Triggers Remodeling of Photosynthetic Protein Complexes. **Plant Physiol.** 168, 615–634 Pubmed

- Hochmal, A. K., Zinzius, K., Charoenwattanasatien, R., Gäbelein, P., Mutoh, R., Tanaka, H., Schulze, S., Liu, G., Scholz, M., Nordhues, A., Offenborn, J. N., Petroutsos, D., Finazzi, G., Fufezan, C., Huang, K., Kurisu, G., and Hippler, M. (2016) Calredoxin represents a novel type of calcium-dependent sensor-responder connected to redox regulation in the chloroplast. **Nat. Commun.** 7, 11847 Pubmed

- Pfannmüller, A., Leufken, J., Studt, L., Michielse, C. B., Sieber, C. M. K., Güldener, U., Hawat, S., Hippler, M., Fufezan, C., and Tudzynski, B. (2017) Comparative transcriptome and proteome analysis reveals a global impact of the nitrogen regulators AreA and AreB on secondary metabolism in Fusarium fujikuroi. PLoS One in press, 1–27 Pubmed

1.12 Contact information

Please refer to:

Dr. Christian Fufezan
Cellzome
Molecular Discovery Research
GlaxoSmithKline
69117 Heidelberg
Germany
eMail: christian@fufezan.net

CHAPTER 2

Quick start

2.1 Download and installation

Please *Download* and install pyQms following these *Installation* instructions. Please consider using a virtual environment (e.g. using the excellent virtualenvwrapper) for using and developing pyQms.

2.2 Matching a peak list

Let's start with a most simple example: Mathing a single peptide on a predefined peak list. Start a Python (3.4+) console and start quantifying in 4 steps:

First import pyQms:

```
import pyqms
```

Second, initialize a isotopologue library (*pyqms.IsotopologueLibrary*) using 'DDSPDLPK' as the example peptide (from BSA example file) and the charge state 2:

```
lib = pyqms.IsotopologueLibrary(
    molecules  = [ 'DDSPDLPK' ],
    charges    = [ 2 ],
)
```

Third, match the library on the provided peak list. You can find a peak list here, which will produce a match with this peptide. Copy and paste the peak list into the Python console.

Fourth, use the *pyqms.IsotopologueLibrary.match_all()* function to quantify the peptide using the peak list:

```
results = lib.match_all(
    mz_i_list = peak_list,
    file_name = 'test',
```

(continues on next page)

(continued from previous page)

```
    spec_id    = 1165,
    spec_rt    = 29.10,
    results    = None
)
```

Done! The peptide has been quantified in the given peak list. Please continue with the next section to learn how to access and process the results.

Note: The keyword arguments *file_name*, *spec_id* and *spec_rt* are hardcoded in this example case. In the advanced examples these information (as well as the peak list) are parsed from the mzML file directly.

2.3 Access and interpret the results

The results object represents the *pyqms.Results* class and is organized as a dictionary:

```
results.keys()
```

Will give the following output:

```
dict_keys(
    [
        m_key(
            file_name='test',
            formula='C(37)H(59)N(9)O(16)',
            charge=2,
            label_percentiles=(('N', '0.000'),)
        )
    ]
)
```

The keys of the *pyqms.Results* class are `namedtuple()` with the following field_names:

- file_name
- formula
- charge
- label_percentiles

file_name related to the original file name of the LC-MS/MS runs, *formula* is the molecular formula of the input molecule/peptide, *charge* refers to the charge state of the matched isotope envelope and *label_percentile* indicates the labeling of the molecule. Default behaviour is to use the natural abundance of the element isotopes (default this fieldname is set to 0% artificical enrichment of nitrogen i.e. ('N','0.000') in a `tuple` of multiple possible labeling percentiles i.e. (('N','0.000'),).

Note: Every input molecule (e.g. peptide 'DDSPDLPK') will be converted to its molecular formula ('C(37)H(59)N(9)O(16)') in Hill notation by pyQms. To map between the peptide and formula, please use the integrated lookups, i.e. *results.lookup['formula to molecule']* or results.lookup['molecule to formula']'. Please consider, that multiple molecules can have the same formula, therefor e.g. *results.lookup['formula to molecule']['C(37)H(59)N(9)O(16)']* is by default a list.

For each of the keys one will get the following dict:

```
{
    'data': [
        match(
            spec_id=1165,
            rt=29.1,
            score=0.9606609710868856,
            scaling_factor=40.75802642055527,
            peaks=(
                (443.7112735313511, 2517650.0, 1.0, 443.7112648946701, 62091), (444.
↪21248374593875, 1156173.75, 0.4459422196277157, 444.2127374486285, 27689),
                (444.71384916266277, 336326.96875, 0.12958327918547244, 444.
↪7142840859656, 8046),
                (445.21533524843596, 58547.0703125, 0.02805309805863953, 445.
↪21582563050043, 1742)
            )
        )
    ],
    'max_score': 0.9606609710868856,
    'len_data': 1,
    'max_score_index': 0
}
```

The keys on the top level of this dictionary are:

- data
- max_score
- len_data
- max_score_index

While *len_data* will indicate how many spectra were matched for the formula in the repective key, *max_score* and *max_score_index* provides the maximum score, which was obtained during matching and the index of this match in the *data* list, respectively. The *data* list contains matches for all single spectra as `namedtuple()`. The following fieldnames are contained in each *match*:

- spec_id
- rt
- score
- scaling_factor
- peaks

Besides the given input information on the spectrum like the spectrum ID (*spec_id*) and the retention time (*spec_rt*) the mScore of the match is provided (*score*) as well as the determined amount/intensity of the molecule in the spectrum (*scaling_factor*). Furthermore, detailed match information are given in *peaks*. This tuple contains for each peak of the isotopologue the following information in this order:

- The measured (and matched) m/z value of the isotope peak in the spectrum
- The measured intensity of the isotope peak in the spectrum
- The relative intensity of the isotopologue peak to the monoisotopic peak
- The calculated m/z value of the isotope peak of the input molecule
- The calculated intensity of the isotope peak of the input molecule

These information can be processed to further analyze, besides the mScore, the quality of the *match*.

Note: Please note, that measured m/z entry in *peaks* can be *None*, if this peak was not found in the input data.

We have now seen, how peptides/molecules can be quantified and how the results can be accessed.

Note: The *pyqms.Results* class offers several functions to access, process and visualize the data. E.g. *pyqms.Results.extract_results()* provides and iterator yielding *key*, *i*, *entry*. The *key* is the `namedtuple()` containing the molecules information, *i* is the position of *entry* in results[key]['data'] and *entry* is the *match* `namedtuple()`.

2.4 Quantify peptides in a whole LC-MS run

This part will describe how to process a whole LC-MS/MS run and quantify multiple peptides in one batch. This example assumes you have started your Python console in the pyqms base folder.

For this example we will use pymzML, which is used to parse mzML files and retrieve the spectra and meta data used for quantification. pymzML will be installed as a requirement (See: *Installation*).

We start again by importing pyQms and initializing a isotopologue library (*pyqms.IsotopologueLibrary*):

```
import pyqms
lib = pyqms.IsotopologueLibrary(
    molecules        = [
        'HLVDEPQNLIK',
        'YICDNQDTISSK',
        'DLGEEHFK'
    ],
    charges          = [2, 3, 4, 5],
)
```

We need to import pymzML and initialize the run. Note, that the path to the BSA1 mzML file ('data/BSA1.mzML') may have to be adjusted. This file can be downloaded using this example script *get_example_BSA_file* (See: *Get the BSA example mzML file*) and can then be found under the 'data' folder in the pyqms base folder.

```
import pymzml
run = pymzml.run.Reader( 'data/BSA1.mzML' )
```

We now iterate over the spectra in the mzML file and quantify all peptides in all MS1 spectra. Before we start the loop we set the results variable to *None*. Please note, that the *results* variable is iteratively passed to *pyqms.IsotopologueLibrary.match_all()*. This will lead to one *results* object, which combines quantifications for all peptides in every spectra. See also description above (see: *access results*) or refer directly to the *pyqms.Results*: class :

```
results = None
for spectrum in run:
    scan_time = spectrum['scan time']
    spec_id   = spectrum['id']
    if spectrum['ms level'] == 1:
        results = lib.match_all(
            mz_i_list = spectrum.centroidedPeaks,
            file_name = 'BSA1',
            spec_id   = spec_id,
            spec_rt   = scan_time,
```

(continues on next page)

(continued from previous page)

```
            results    = results
        )
```

Note: pymzML centroids spectra if these are not already centroided, if *spectrum.centroidedPeaks* is accessed.

The *results* can now be accessed as described above (see: *access results*). Furthermore the *pyqms.Results* class can be pickled:

```
import pickle
pickle.dump(
    results,
    open(
        'data/BSA1_pyQms_results.pkl',
        'wb'
    )
)
```

For further examples and how to use the adaptor functions, please refer to the next section.

2.5 Use the adaptors, Luke

The *Adaptors* functions are useful for parsing a set of identified peptides (e.g. from Ursgal result files; Ursgal documentation) including retention time information for determining the maximum intensity of every (identified) peptide in the LC-MS/MS measurement. Furthermore, adaptors can be added to e.g. read results of other analysis pipelines and tools.

The current adaptor to read Ursgal results can be used as follows for the shipped identification result file of the database search engine OMSSA. Please note, that if the adaptors are used one need to define fixed modifications like Carbamidomathylation as presented. This modification and the molecules will then be correctly formatted as input for pyqms:

```
import pyqms
import pyqms.adaptors
input_fixed_labels = {
    'C' : [
        {
            'element_composition' : {
                'O'   : 1,
                'H'   : 3,
                '14N' : 1,
                'C'   : 2
            },
            'evidence_mod_name': 'Carbamidomethyl'
        },
    ]
}
formatted_fixed_labels, evidence_lookup, molecules = pyqms.adaptors.parse_evidence(
    fixed_labels   = input_fixed_labels,
    evidence_files = [ 'data/BSA1_omssa_2_1_9_unified.csv' ],
)
```

The returned objects can be used a direct input for the pyQms *pyqms.IsotopologueLibrary*. The advantage of parsing evidence files is, that MS2 identification information is added to the results and can e.g. be used for defining

RT windows for a correct quantification of every peptide:

```
lib = pyqms.IsotopologueLibrary(
    molecules    = molecules,
    charges      = [1, 2, 3, 4, 5],
    fixed_labels = formatted_fixed_labels,
    evidences    = evidence_lookup
)
```

2.6 Further examples and more adavanced usage

Please refer to the *Example Scripts* section for more usage examples and ready-to-go Python scripts for quantification, data analysis and visualization.

CHAPTER 3

Contents

3.1 Isotopologue Library

class pyqms.**IsotopologueLibrary**(*molecules=None*, *charges=None*, *metabolic_labels=None*, *fixed_labels=None*, *params=None*, *trivial_names=None*, *verbose=True*, *evidences=None*)

The Isotopologue library is the core of pyQms.

Keyword Arguments

- **molecules** (*list of str*) – Molecules used to build the library, for more details see below.
- **charges** (*list of int*) – Charge list used to build the library
- **metabolic_labels** (*dict*) – see below
- **fixed_labels** (*dict*) – see below
- **params** (*dict*) – Match parameters, see *pyqms.params*
- **trivial_names** (*dict*) – Dictionary that is used to build up lookups. Key is a molecule and value a trivial name.
- **evidences** (*dict*) – Dictionary that is used to build up additional lookups. Key is a formula pointing to a subdict. Subdict has molecules as keys and values are 'trivial_names' as a list and 'evidences' holding evidence/identification information
- **verbose** (*bool*) – Be verbose or not during initialization and matching.

Keyword argument examples:

- **molecules** The molecule format can be anything that the ChemicalComposition class understands. Currently this can for example be:

```
[
    '+{0}'.format('H2O'),
    '{peptide}'.format(peptide='PEPTIDE'),
```

(continues on next page)

(continued from previous page)

```
    '{peptide}+{0}'.format('PO3', peptide='PEPTIDE'),
    '{peptide}#{unimod}:{pos}'.format(
        peptide = peptide,
        unimod = 'Oxidation',
        pos = 1
    )
]
```

- **metabolic_labels** is used to define new element pools with enriched isotopes. The dict key defines an enriched element, e.g. 15N or 13C and its value is a list of floats [0 - 1.0] defining enrichment.The combination of those pools is used to calculate isotopologues:

```
{
    '15N' : [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99]
}
```

- **fixed_labels** are based on unimod. Fixed molecules do not change the shape of the isotoplogue drastically but introduce a simple mass shift, like SILAC, 18O or others.

  The format is for example:

```
{
  'R' : ['C(-6) 13C(6) N(-4) 15N(4)','']
}
```

  Returns Isotopologue library as dict where the top key is always the chemical forumla unimod style.

  Return type dict

simplified:

```
{
    'C(34)H(53)N(7)O(15)': {
        'cc': {
            'C': 34, 'H': 53, 'N': 7, 'O': 15
        },
        'env': {
            (('N', '0.000'),): {
                # charge :
                1: {
                    # all transformed mz values
                    'atmzs': {
                        800443,
                        800444,
                        # ... skipped
                        803459,
                        803460,
                        803461
                    },
                    # theoretical mz values
                    'mz': [
                        800.4472772254203,
                        801.450389542063,
                        802.4536114854914,
                        803.4568170275203,
                        804.4597382487398,
```

(continues on next page)

(continued from previous page)

```
                    805.463171867346,
                    806.4631454917885,
                    807.4676949759603
                ],
                # transformed mz values within error
                # packages are on on peak level
                'tmzs': [
                    {
                        800443,
                        800444,
                        # ... skipped
                        800451
                    },
                    {
                        801446,
                        801447,
                        # ... skipped
                        801454
                    },
                    {
                        802450,
                        802451,
                        # ... skipped
                        802458
                    },
                    {
                    803453,
                        803454,
                        803455,
                        # ... skipped
                        803461
                    },
                    None,
                    None,
                    None,
                    None
                ]
            },
            # charge independent information
             'abun': [
                64799,
                26251,
                7164,
                1456,
                175,
                20,
                1,
                0
            ],
            'c_peak_pos': [
                0,
                1,
                2,
                3,
                None,
                None,
                None,
```

(continues on next page)

(continued from previous page)


```
                    None
                ],
                'isot': [],
                'mass': [
                    799.3599640346001,
                    800.3629760500413,
                    801.3660976813065,
                    802.3692029128123,
                    803.3720238519379,
                    804.3753571372156,
                    805.3752307742944,
                    806.3796798135622
                ],
                'n_c_peaks': 4.0,
                'relabun': [
                    1.0,
                    0.40511743373159037,
                    0.11054965400744385,
                    0.022466784140529883,
                    0.002693331560888158,
                    0.0003019650321460501,
                    7.716705830708012e-06,
                    3.639837831552297e-08
                ]
            }
        }
    }
}
```


match_all(*mz_i_list=None*, *file_name=None*, *spec_id=None*, *spec_rt=None*, *results=None*)

Matches all isotopologues in the library agains a given *mz_i_list*

Parameters

- **mz_i_list** (*list of tuples*) – Spectrum information that should be matched against. Tuples of m/z and intensity
- **file_name** (*str*) – Information used for storage purpose. Useful if multiple files are parsed with one pyqms.result instance.
- **spec_id** (*int*) – Information used for storage purpose.
- **spec_rt** (*float*) – Information used for storage purpose.
- **results** (*pyqms.Results*) – (optional)

If a results object is passed to match_all, then this object will be updated and returned. This is for e.g. to accumulate results for a whole LC-MS/MS run.

For various examples using match_all please refer to the example scripts.

Returns Object holding all quantitative information

Return type results class object (obj)

match_isotopologue(*index=None*, *formula=None*, *charge=None*, *label_percentile=None*, *spec_tmz_set=None*, *spec_tmz_lookup=None*, *mz_i_list=None*, *mz_score_percentile=None*)

Matches a single isotopologue onto a *mz_i_list* or *spec_tmz_set*

Parameters

- **index** (*int*) – Using this index one can retrieve all information about the molecule, i.e. lower_mz, upper_mz, charge, label_percentile, formula from *self.formulas_sorted_by_mz*. Alternatively, one can use the more verbose option: formula, charge and label_percentile
- **formula** (*str*) – pyqms formula type
- **charge** (*int*) – molecule charge
- **label_percentile** – pyQms label percentile
- **mz_i_list** (*list of tuples*) – List of m/z and intensity tuples, will be transformed to a spec_tmz_set given the defined precession. Alternatively, spec_tmz_set can be used as input.
- **spec_tmz_set** (*set of ints*) – tmz value set used for matching. Requires spec_tmz_lookup to get the actual mz which is required for scoring.
- **mz_score_percentile** (*float*) – Weighting of mz used for scoring. (1 - mz_score_percentile) is then intensity weighting. Values 0 - 1.0.

Note: Depending on the machine (some measure intensity better than others) adjusting mz_score_percentile value will give more accurate results. Best adjusted in pyqms.params (which can be passed during isotoplogue lib initialization)

Returns

Match results (tuple of score, scaling factor and matched peaks).

- **score** reflects the fit of the theoretical isotopologue to the measured (both mz and intensities are compared)
- **scaling factor** reflects the actual amount of the molecule in the respective spectrum. It is defined as the sum of the total measured intensities divided by the sum of the total calculated intensities
- **matched_peaks** is list of tuples that contain measured_mz, measured_i, rel_i, calculated_mz, calculated_i

Multiple m/z values can occur in the range of the measured precision of every peak of the isotopologue, thus all combinations are considered and scored. Only the best scored match is returned for each isotopologue.

print_overview(*formula*, *charge=None*)

Prints an overview of a given molecule or formula to the std.out

Parameters

- **formula** (*str*) – Either formula or molecule
- **charge** (*int*) – Charge of the molecule

Examples

For PEPTIDE and charge 1:

```
Chemical formula C(34)H(53)N(7)O(15)
(('N', '0.000'),)
Isotope                                                Abundance
```

(continues on next page)

(continued from previous page)

pos	Mass	m/z [MH]+1	transformed	rel.	
0	799.3599640346	800.4472772254	64799	1.00000000000	0
1	800.3629760500	801.4503895421	26251	0.40511743373	1
2	801.3660976813	802.4536114855	7164	0.11054965401	2
3	802.3692029128	803.4568170275	1456	0.02246678414	3
4	803.3720238519	804.4597382487	175	0.00269333156	None
5	804.3753571372	805.4631718673	20	0.00030196503	None
6	805.3752307743	806.4631454918	1	0.00000771671	None
7	806.3796798136	807.4676949760	0	0.00000003640	None

score_matches(*matched_peaks*, *mz_score_percentile*)

Score matched peaks.

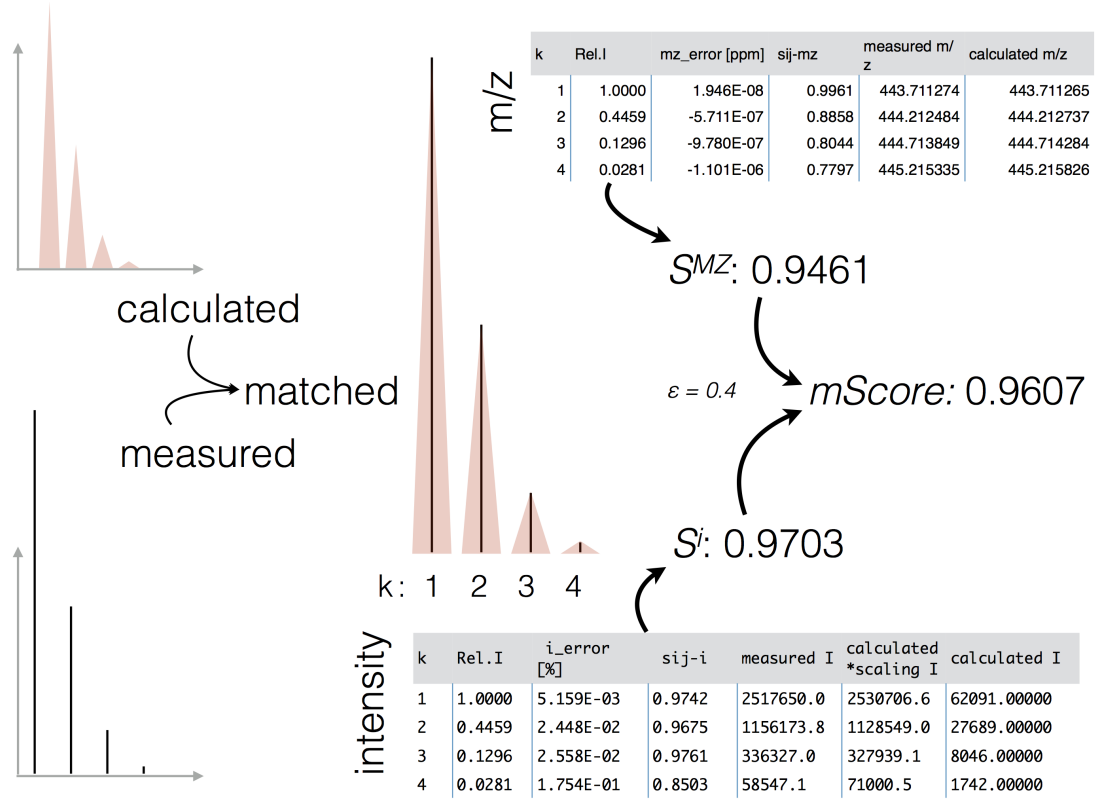
Parameters

- **matched_peaks** (*list of tuples*) – List of tuples containing
  - measured_mz (mmz)
  - measured_intensity (mi)
  - relative_intensity_of_calculated_isotopologue_peak (ri)
  - calculated_mz (cmz)
  - calculated_i (ci)
- **mz_score_percentile** (*float*) – weighting of mz score

Parameters that influence the scoring are 'MIN_REL_PEAK_INTENSITY_FOR_MATCHING'

Example plots

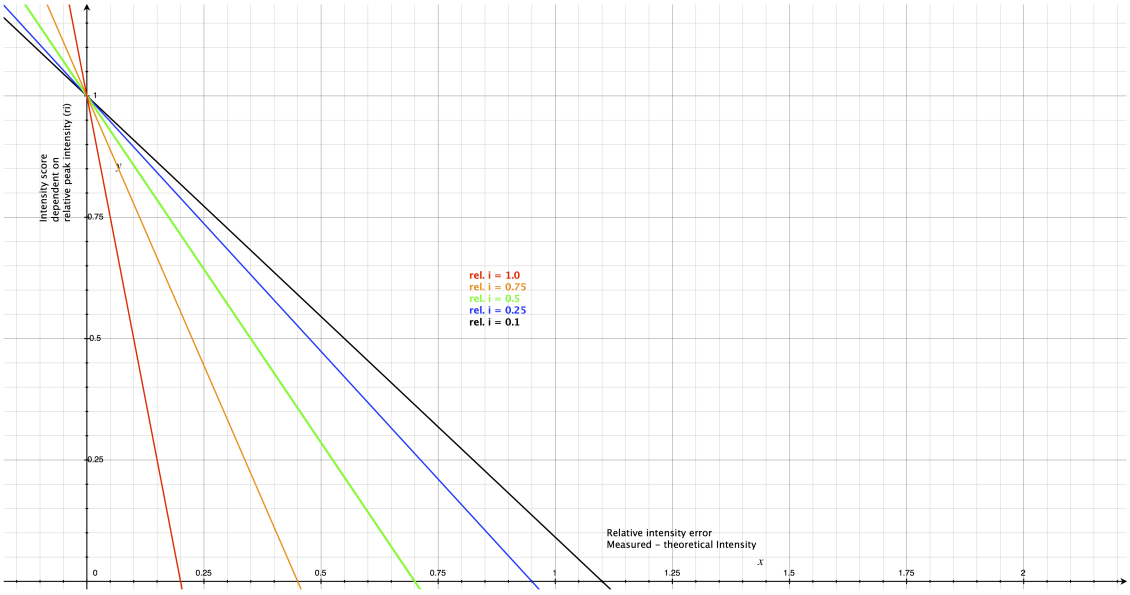
The figure below highlights the scoring principle. Erros for m/z and intensity values are determined and combined into the final mScore. For each peak of the isotopologue both errors are determined and influence the final score.



k	Rel.I	mz_error [ppm]	sij-mz	measured m/z	calculated m/z
1	1.0000	1.946E-08	0.9961	443.711274	443.711265
2	0.4459	-5.711E-07	0.8858	444.212484	444.212737
3	0.1296	-9.780E-07	0.8044	444.713849	444.714284
4	0.0281	-1.101E-06	0.7797	445.215335	445.215826

k	Rel.I	i_error [%]	sij-i	measured I	calculated *scaling I	calculated I
1	1.0000	5.159E-03	0.9742	2517650.0	2530706.6	62091.00000
2	0.4459	2.448E-02	0.9675	1156173.8	1128549.0	27689.00000
3	0.1296	2.558E-02	0.9761	336327.0	327939.1	8046.00000
4	0.0281	1.754E-01	0.8503	58547.1	71000.5	1742.00000

Calculated intensities are scaled to match the measured value and the deviation is calculated. The lower the intensity, the less accurate teh actual peaks are represented. To compensate for this, the intensity score decreases faster for large relative intensities compared to small relative intensities. This is highlighted in the following figure. Legend, x-axis represents the relative intensity error (measured - theoretical intensity) and the y-axis the intensity score. Different colors represent various relative peak intensities.



Scoring

Note: The proper display of the formulas of the next section requires access to the Internet when browsing the HTML documentation. The formulas are correctly embedded into the pdf of the documentation.

The pyQms matching score (mScore) is based on the work of Gower (1971) *A General Coefficient of Similarity and Some of Its Properties*, Biometrics (27), 857-871. The matching and scoring is performed on the m/z values and the intensity values independently yielding two scores, i.e. S^{mz} and $S^{intensity}$. In both cases, each peak k is scored, comparing the measured value i with the calculated value j (equation 1), whereas a perfect match is 1. Each peak of the isotopologue that has a relative intensity (relative to the maximum intensity isotope peak) r_k above the matching threshold (by default 1% of the maximum intensity isotope peak) is matched and scored.

$$s_{ijk} \in [0, 1] \tag{3.1}$$

The m/z score

For each peak k, the m/z similarity between measured value i and the calculated value j is defined as

$$s_{ijk}^{mz} = 1 - (\frac{\delta_{ijk}^{mz}}{\alpha}) \tag{3.2}$$

Whereas $delta_{ijk}^{mz}$ the difference in ppm between measured mz_{ik} and calculated mz_{jk} and α defines the range in ppm, in which the score decreases from 1 to 0 in a linear fashion. In principle, α is equal to the precision of the measurement defined by the user (pyQms parameter "REL_MZ_RANGE", default 5 ppm, http://pyqms.readthedocs.io/en/latest/params.html). For example, if the difference between measured and theoretical m/z values would be 2.5 ppm, then the s_{ijk}^{mz} score for this peak k would be 0.5.

The total m/z score for all peaks termed S^{mz} is the weighted sum of all single similarity m/z scores s_{ijk}^{mz} (equation 3). The weighting is defined by the theoretical intensity of the peak k relative to the highest peak in the theoretical isotope pattern, termed r_k.

$$S^{mz} = \frac{\sum^{k} s_{ijk}^{mz} r_k}{\sum^{k} r_k} \tag{3.3}$$

The intensity score

Prior to intensity scoring, the scaling factor σ is calculated by comparing the intensities of the measured i and calculated j intensities for all peaks k within the matching threshold (see above). This scaling factor is calculated by dividing the weighted sum of the measured intensity by the weighted sum of the theoretical intensities (equation 4).

$$\sigma = \frac{\sum^{k} intensity_{ik} r_k}{\sum^{k} intensity_{jk} r_k} \tag{3.4}$$

Using this scaling factor, which is equal to the abundance of the measured molecule, one can calculate $\delta_{ijk}^{intensity}$, which is the relative intensity error between measured and theoretical intensity for each peak k (equation 5).

$$\delta_{ijk}^{intensity} = \frac{|intensity_{ik} - \sigma intensity_{jk}|}{\sigma intensity_{jk}} \tag{3.5}$$

The intensity score of peak k is then defined (equation 6).

$$s_{ijk}^{intensity} = 1 - (\frac{\delta_{ijk}^{intensity}}{1 - r_k + \epsilon}) \tag{3.6}$$

In analogy to the m/z score (s_{ijk}^{mz}), the denominator defines the range in which the peak based intensity score decreases from 1 to 0. However, in contrast to the m/z score, the intensity error has to be weighted by the abundance of each peak (1 - r_k) as more abundant peaks can be measured more accurately than smaller peaks. Additionally, we introduced (pyQms parameter "REL_I_RANGE", default 0.2), which represents the most conservative relative error applied to the most precisely measured peak (r_k = 1). Thus, the overall relative error (denominator) will increase with lower peaks The total intensity score $S^{intensity}$ is the weighted sum of all similarity scores k in analogy to the S^{mz} score:

$$S^{intensity} = \frac{\sum^{k} s_{ijk}^{intensity} r_k}{\sum^{k} r_k} \tag{3.7}$$

The combined final score: mScore The final score is termed mScore and is a sum of S^{mz} and $S^{intensity}$. However, because some machines can measure m/z much more accurately then intensities, we introduced ξ to allow for flexibilities depending on the type of mass spectrometer used. ξ (the pyQms parameter "MZ_SCORE_PERCENTILE", default 0.4) is the fraction the S^{mz} score is weighted into the sum. Thus, the final mScore is defined as:

$$mScore = \xi S^{mz} + (1 - \xi) S^{intensity} \tag{3.8}$$

3.2 Result Class

class pyqms.**Results**(*lookup=None*, *params=None*, *fixed_labels=None*, *metabolic_labels=None*, *aa_compositions=None*, *isotopic_distributions=None*, *charges=None*, *verbose=False*)

pyQms results class.

Holds all matching information and lookups. Can be accessed as a dictionary. Several lookup allow the mapping of moleclar formulas to molecules (e.g. peptides) and/or trivial names (e.g. protein names).

Structure

key (named tuple)

- file_name
- formula
- charge
- label_percentiles

value (named tuple)

- spec_id
- rt
- score
- scaling_factor
- peaks

add(*key*, *value*)

Adds match to the result container.

Parameters

- **key** (*named tuple*) – (file_name, formula, charge, label_percentiles)

- **value** (*named tuple*) – (spec_id, rt, score, scaling_factor, peaks)

Returns formatted_key (named tuple) : (file_name, formula, charge, label_percentiles)

Structure

key (named tuple)

- file_name
- formula
- charge
- label_percentiles

value (named tuple)

- spec_id
- rt
- score
- scaling_factor
- peaks

calc_amounts_from_rt_info_file(*rt_info_file=None*, *rt_border_tolerance=None*, *calc_amount_function=None*, *evidence_score_field='PEP'*, *buffer_only=False*, *buffered_csv_dicts=None*)

Function to calculate molecule/peptide amounts based on the quant summary/rt info file genearte by *write_rt_info_file()*. See e.g. example script generate_quant_summary_file.py. A function to calculate the final molecule amounts can be defined otherwise the default maximum intensity function is used.

Parameters

- **rt_info_file** (*str*) – output file name of the quant summary/rt info csv file, must be be a complete or relative path
- **rt_border_tolerance** (*int*) – retention time border tolerance in minutes
- **calc_amount_function** (*obj*) – python function to calculate final amounts based on a simple dictionary structure

The function to calculate the amount of the molecules (calc_amount_function) should be able to process the below shown dictionary structure (obj_for_calc_amount). The default function returns the maximum amount in the retention time window or in the complete profile. The function should return the determined amount, the retention time (or approximate) as well as the score. If functions are used which determine the amount over more than one spectrum retention times for this amount should be e.g. at the maximum intensity of the profile. Scores could be e.g. averaged or also the score at the maximum amount could be used.

Example for obj_for_calc_amount:

```
{
    'rt'       : [rt1,rt2,...],
    'i'        : [in1,in2,...],
    'scores'   : [sc1,sc2,...],
    'spec_ids' : [id1,idt2,...],
}
```

Example key names (default):

- 'max I in window'
- 'max I in window (rt)'
- 'max I in window (score)'
- 'auc in window' (area under curve)
- 'sum I in window' (summed up intensities)

curate_rt_windows (*evidence_dict*, *rt_tolerance*)
Internal function to curate RT windows

determine_max_itensity (*obj_for_calc_amount*)
Function to determine the maximum intensity in given elution profile. The structure of the object passed to the function is shown below. This fucntion can be used as a template function to write and use of function to determine the amount of a molecule e.g. area under curve or summed up intensities. All self written function must return amount, rt at or around the amount and the mScore at or around the amount in an dictionary with appropiate key names.

Example key names (default):

- 'max I in window'
- 'max I in window (rt)'
- 'max I in window (score)'

Note: This is the default function to determine the peptide amount when write_amount_csv() is called.

Examples:

```
{
   'rt'       : [rt1,rt2,...],
   'i'        : [in1,in2,...],
   'scores'   : [sc1,sc2,...],
   'spec_ids' : [id1,idt2,...],
}
```

Returns keys are shown above

Return type dict

extract_results (*molecules=None*, *charges=None*, *file_names=None*, *label_percentiles=None*, *formulas=None*, *score_threshold=None*)
Extract selected results.

Extracts all matches from the results instance that meet given filter criteria.

Parameters

- **molecules** (*list of str, optional*) – considered molecules. Those will be translated using self._translate_molecules_to_formulas()
- **charges** (*list of int, optional*) – considered charge states.
- **file_names** (*list of str, optional*) – list of file names to be considered.
- **label_percentiles** (*list of tuple, optional*) – list of label percentile tuples to be considered.

- **formulas** (*list of str*) – list of chemical formulas

Yields *key, i, entry (tuple)* – result class key, index of entry and entry

Structure

key (named tuple)

- file_name
- formula
- charge
- label_percentiles

value (named tuple)

- spec_id
- rt
- score
- scaling_factor
- peaks

max_score(*molecules=None*, *charges=None*, *file_names=None*, *label_percentiles=None*, *formulas=None*)

Find max score for a given set of parameters.

Parameters

- **molecules** (*list of str, optional*) – considered molecules. Those will be translated using self._translate_molecules_to_formulas()
- **charges** (*list of int, optional*) – considered charge states.
- **file_names** (*list of str, optional*) – list of file names to be considered.
- **label_percentiles** (*list of tuple, optional*) – list of label percentile tuples to be considered.
- **formulas** (*list of str*) – list of chemical formulas

Returns key is appropriate key in result.dict

Return type best_score, key, index (tuple)

plot_MIC_3D(*key*, *file_name=None*, *rt_window=None*, *i_transform=None*)

Plot MIC from results using rpy2 in 3D.

plot_MICs_2D(*key_list*, *file_name=None*, *rt_window=None*, *i_transform=None*, *xlimits=None*, *additional_legends=None*, *title=None*, *zlimits=None*, *ablines=None*, *graphics=None*)

Parameters **additional_legends** (*dict*) – key points on lists of strings that are plotted as well.

write_result_csv(*output_file_name=None*)

Write raw results into a .csv file

Parameters **output_file_name** (*str*) – output file name of the csv containing containing all raw results, should be a complete path

Warning: Depending on data size the resulting csv can become very large. Some csv viewer can not handle files with a large number of lines.

Keys in csv:

- Formula : molecular formula of the molecule (str)
- Molecule : molecule or trivial name (str)
- Charge : charge of the molecule (int)
- ScanID : ScanID of the quantified spectrum (int)
- Label Percentiles : Labeling percentile ((element, enrichment in %),)
- Amount : the determined amount of the molecule
- Retention Time : retetention time of the ScanID
- mScore : score of the isotopologue match
- Filename : filename of spectrum input files

write_result_mztab(*output_file_name=None*, *rt_border_tolerance=None*)

Write minimal peptide quantification results into a .mztab file. It is neccessary to specify the 'formula to evidences' dict in the lookup of the results class to write results!

Note:

This basic mzTab writer is still in beta stage. Use and evaluate with care.

PRIDE CV based quantifcation unit and value is fixd to:

- PRIDE:0000393, Relative quantification unit
- PRIDE:0000425, MS1 intensity based label-free quantification method

Args:

output_file_name (str): output file name of the mztab containing containing all raw results, should be a complete path

Note:

Adiitional information has to be passed tot he result class for a more complete mztab output.

Keys in mztab:

- sequence
- accession
- unique
- database
- database_version
- search_engine
- best_search_engine_score[1-n]
- modifications
- retention_time

- retention_time_window
- charge
- mass_to_charge
- peptide_abundance_study_variable[1-n]
- peptide_abundance_stdev_study_variable[1-n]
- peptide_abundance_std_error_study_variable[1-n]
- search_engine_score[1-n]_ms_run[1-n]
- peptide_abundance_assay[1-n]
- spectra_ref
- opt_{identifier}_*
- reliability
- uri

Addtional information can be added to the mzTab file by adding a dict like shown below to the results.lookup dict under the key 'mztab_meta_info'.:

```
mztab_meta_info = {
    'protein_search_engine_score'    : [],
    'psm_search_engine_score'        : ['[MS,MS:1001475,OMSSA:evalue, ]'],
    'fixed_mod'                      : ['[UNIMOD, UNIMOD:4, Carbamidomethyl, ]
↪'],
    'variable_mod'                   : ['[UNIMOD, UNIMOD:35, Oxidation, ]'],
    'study_variable-description'     : ['Standard BSA measurement'],
    'ms_run-location'                : ['BSA1.mzML'],
}
```

write_rt_info_file(*output_file=None*, *list_of_csvdicts=None*, *trivial_name_lookup=None*, *rt_border_tolerance=None*, *update=True*, *buffer_only=False*)

Function to write a default quant summary/rt info file. See e.g. example script generate_quant_summary_file.py.

Parameters

- **output_file** (*str*) – output file name of the csv, should be a complete path
- **list_of_csvdicts** (*list*) – list of dictionaries passed to the DictWriter class, default fieldnames can be found below
- **trivial_name_lookup** (*dict*) – self defined trivial_name_lookup, see format below.
- **rt_border_tolerance** (*int*) – retention time border tolerance in minutes
- **update** (*bool*) – if True read in or passed dictionaries in list_of_csvdicts will be updated with default evidence and trivial name information

The quant summary file can manually be updated (e.g. the start and stop RT information). If an evidence lookup is present in the result class (can be passed to the isotopologue library or later be set in the result class), these information are used to define the retention time borders (e.g. peptide identfication information from peptide spectrum matches).

Default fieldnames:

- file_name : filename of spectrum input file

- formula : molecular formula of the molecule
- molecule : molecule or trivial name
- trivial_name(s) : protein or trivial names
- label_percentiles : labeling percentile ((element, enrichment in %),)
- charge : charge of the molecule
- start (min) : start of retention time window
- stop (min) : stop of retention time window
- max I in window : maximum intensity in retention time window
- max I in window (rt) : retention time @ maximum intensity in retention time window
- max I in window (score) : score @ maximum intensity in retention time window
- auc in window : area under curve in retention time window
- sum I in window : summed up intensities in retention time window
- evidences (min) : all evidences/identifications (score@rt;...)

Trivial name lookup example:

```
{
    'C(33)H(59)14N(1)N(8)O(9)S(1)' : ['BSA','Bovine serum albumine']
}
```

3.3 Chemical composition

class pyqms.**ChemicalComposition**(*sequence=None*, *aa_compositions=None*, *isotopic_distributions=None*)

Chemical composition class. The actual sequence or formula can be reset using the *add* function.

Keyword Arguments

- **sequence** (*str*) – Peptide or chemical formula sequence
- **aa_compositions** (*Optional[dict]*) – amino acid compositions
- **isotopic_distributions** (*Optional[dict]*) – isotopic distributions

Keyword argument examples:

- **sequence**

This can for example be:

```
molecules = [
    '+H2O2H2-OH',
    '+{0}'.format('H2O'),
    '{peptide}'.format(pepitde='ELVISLIVES'),
    '{peptide}+{0}'.format('PO3', peptide='ELVISLIVES'),
    '{peptide}#{unimod}:{pos}'.format(
        peptide = 'ELVISLIVES',
        unimod = 'Oxidation',
        pos = 1
    )
]
```

Examples

```
>>> c = pyqms.ChemicalComposition()
>>> c.use("ELVISLIVES#Acetyl:1")
>>> c.hill_notation()
'C52H90N10O18'
>>> c.hill_notation_unimod()
'C(52)H(90)N(10)O(18)'
>>> c
{'O': 18, 'H': 90, 'C': 52, 'N': 10}
>>> c.composition_of_mod_at_pos[1]
defaultdict(<class 'int'>, {'O': 1, 'H': 2, 'C': 2})
>>> c.composition_of_aa_at_pos[1]
{'O': 3, 'H': 7, 'C': 5, 'N': 1}
>>> c.composition_at_pos[1]
defaultdict(<class 'int'>, {'O': 4, 'H': 9, 'C': 7, 'N': 1})
```

```
>>> c = pyqms.ChemicalComposition('+H2O2H2')
>>> c
{'O': 2, 'H': 4}
>>> c.subtract_chemical_formula('H3')
>>> c
{'O': 2, 'H': 1}
```

Note: We did not include mass calculation, since pyQms will do it much more accurately using unimod and other element enrichments.

---

add_chemical_formula(*chemical_formula*)

Adds chemical formula to the instance

Chemical formula can be a string or a dictionary with the element count.

For example:

```
chemical_formula = 'C18H36N9O18'
chemical_formula = {
    'C' : 18,
    'H' : 36,
    'N' : 9,
    'O' : 18
}
```

add_peptide(*peptide*)

Adds peptide sequence to the instance.

Note: Only standard amino acids can be processed. If one uses special amino acids like (U of F) they have to be added to knowledge_base.py.

---

clear()

Resets all lookup dictionaries and self

One class instance can be used analysing a series of sequences, thereby avoiding class instantiation overhead.

> **Warning:** Make sure to reset when looping over sequences and use the class. Chemical formulas (elemental compositions) will accumulate if not resetted.

composition_at_pos = None
chemical composition at given peptide position incl modifications (if peptide sequence was used as input or using the *use* function)

> **Note:** Numbering starts at position 1, since all PSM search engines use this nomenclature.

Type dict

composition_of_aa_at_pos = None
chemical composition of amino acid at given peptide position (if peptide sequence was used as input or using the *use* function)

> **Note:** Numbering starts at position 1, since all PSM search engines use this nomenclature.

Examples:

```
c.composition_of_mod_at_pos[1] = {
    '15N': 2, '13C': 6, 'N': -2, 'C': -6
}
```

Type dict

composition_of_mod_at_pos = None
chemical composition of unimod modifications at given position (if peptide sequence was used as input or using the *use* function)

> **Note:** Numbering starts at position 1, since all PSM search engines use this nomenclature.

Type dict

hill_notation(*include_ones=False*, *cc=None*)
Formats chemical composition into Hill notation string.

Parameters **cc** (*dict, optional*) – elemental composition dict

Returns

Hill notation format of self.

For example:

```
'C50H88N10O17'
```

Return type str

hill_notation_unimod(*cc=None*)
Formats chemical composition into Hill notation string adding unimod features.

Parameters **cc** (*dict, optional*) – elemental composition dict

Returns

Hill notation format including unimod format rules of self.

For example:

```
'C(50)H(88)N(10)O(17)'
'C(50)H(88)14N(1)N(9)(17)'
```

Return type str

subtract_chemical_formula(*chemical_formula*)
Subtracts chemical formula from instance.

subtract_peptide(*peptide*)
Subtracts peptide (chemical formula) from instance.

use(*sequence*)
Re-initialize the class with a new sequence

This is helpful if one ones to use the same class instance for multiple sequence since it remove class instantiation overhead.

Parameters **sequence** (*str*) –

Note: Will clear the current chemical composition dict!

3.4 Unimod mapper

class pyqms.**UnimodMapper**
UnimodMapper class that creates lookup to the unimod.xml located in kb/ext/unimod.xml and offers several helper methods.

Mapping from e.g. name to composition or unimod ID to mass is possible.

Please refer to unimod for further informations on modifications including naming, formulas, masses etc.

appMass2element_list(*mass*, *decimal_places=2*)
Creates a list of element composition dicts for a given approximate mass

Parameters **mass** (*float*) – approximate mass of modification

Keyword Arguments **decimal_places** (*int*) – Precision with which the masses in the Unimod is compared to the input, i.e. round(mass, decimal_places)

Returns Dicts of elements

Return type list

Examples:

```
>>> import pyqms
>>> U = pyqms.UnimodMapper()
>>> U.appMass2element_list(18, decimal_places=0)
    [{'F': 1, 'H': -1}, {'13C': 1, 'H': -1, '2H': 3},
    {'H': -2, 'C': -1, 'S': 1}, {'H': 2, 'C': 4, 'O': -2},
    {'H': -2, 'C': -1, 'O': 2}]
```

appMass2id_list (*mass*, *decimal_places=2*)
Creates a list of unimod IDs for a given approximate mass

Parameters **mass** (*float*) – approximate mass of modification

Keyword Arguments **decimal_places** (*int*) – Precision with which the masses in the Unimod is compared to the input, i.e. round(mass, decimal_places)

Returns Unimod IDs

Return type list

Examples:

```
>>> import pyqms
>>> U = pyqms.UnimodMapper()
>>> U.appMass2id_list(18, decimal_places=0)
    ['127', '329', '608', '1079', '1167']
```

appMass2name_list (*mass*, *decimal_places=2*)
Creates a list of unimod names for a given approximate mass

Parameters **mass** (*float*) – approximate mass of modification

Keyword Arguments **decimal_places** (*int*) – Precision with which the masses in the Unimod is compared to the input, i.e. round(mass, decimal_places)

Returns Unimod names

Return type list

Examples:

```
>>> import pyqms
>>> U = pyqms.UnimodMapper()
>>> U.appMass2name_list(18, decimal_places=0)
    ['Fluoro', 'Methyl:2H(3)13C(1)', 'Xle->Met', 'Glu->Phe', 'Pro->Asp']
```

composition2id_list (*composition*)
Converts unimod composition to unimod name list, since a given composition can map to mutiple entries in the XML.

Parameters **composition** (*dict*) – element composition (element, count pairs)

Returns Unimod IDs

Return type list

composition2mass (*composition*)
Converts unimod composition to unimod monoisotopic mass.

Parameters **composition** (*dict*) – element composition (element, count pairs)

Returns monoisotopic mass

Return type float

composition2name_list (*composition*)
Converts unimod composition to unimod name list, since a given composition can map to mutiple entries in the XML.

Parameters **composition** (*dict*) – element composition (element, count pairs)

Returns Unimod names

Return type list

id2composition(*unimod_id*)
Converts unimod ID to unimod composition

Parameters **unimod_id** (*int*) – identifier of modification

Returns Unimod elemental composition

Return type dict

id2mass(*unimod_id*)
Converts unimod ID to unimod mass

Parameters **unimod_id** (*int*) – identifier of modification

Returns Unimod mono isotopic mass

Return type float

id2name(*unimod_id*)
Converts unimod ID to unimod name

Parameters **unimod_id** (*int*) – identifier of modification

Returns Unimod name

Return type str

mass2composition_list(*mass*)
Converts unimod mass to unimod element composition list, since a given mass can map to mutiple entries in the XML.

Parameters **mass** (*float*) – mass of modification

Returns Unimod elemental compositions

Return type list

mass2id_list(*mass*)
Converts unimod mass to unimod name list, since a given mass can map to mutiple entries in the XML.

Parameters **mass** (*float*) – mass of modification

Returns Unimod IDs

Return type list

mass2name_list(*mass*)
Converts unimod mass to unimod name list, since a given mass can map to mutiple entries in the XML.

Parameters **mass** (*float*) – mass of modification

Returns Unimod names

Return type list

name2composition(*unimod_name*)
Converts unimod name to unimod composition

Parameters **unimod_name** (*str*) – name of modification (as named in unimod)

Returns Unimod elemental composition

Return type dict

name2id(*unimod_name*)
Converts unimod name to unimod ID

Parameters **unimod_name** (*str*) – name of modification (as named in unimod)

Returns Unimod ID

Return type int

name2mass(*unimod_name*)
Converts unimod name to unimod mono isotopic mass

Parameters **unimod_name** (*str*) – name of modification (as named in unimod)

Returns Unimod mono isotopic mass

Return type float

name2specificity_site_list(*unimod_name*)
Converts unimod name to list of specified amino acids or sites

Parameters **unimod_name** (*str*) – name of modification (as named in unimod)

Returns list of specificity sites

Return type list

3.5 Adaptors

adaptors.**_parse_evidence_and_format_fixed_labels**(*data=None*)
Reformats input params to pyQms compatible params. Additionally evidence files are read in and the fixed labels are reformatted (stripped from the modifications if peptides are read in) as required by pyQms. This is especially required if data/samples contains Carbamidomethylation as modification and the sample was e.g. 15N labeled. This ensures that the nitrogens pools of the peptides (which are 15N labeled) do not mix up with the nitrogen pool of the Carbamidomethylation (14N since intriduced during sample preparation). All fixed modifications needs to be specified so that is can be ignored from the input evidence file but correctly formatted for the parameters.

Example format:

```
{
    'molecules' : {'PEPTIDEA',...},
    'evidences' : {
        'C18H36O18N9' : {
            'PEPTIDEA' : {
                'evidences' : [
                    {
                        'RT':13.37,
                        'score':0.01,
                        'score_field':'PEP'
                    },
                    ...
                ],
                'trivial_names':[
                    'PROTEIN_NAME',
                    'PATHWAY_NAME',
                    ...
                ]
        },
    },
    'charges'   : {1,2,...},
    'params'    : {
```

(continues on next page)

(continued from previous page)

```
        'MACHINE_OFFSET_IN_PPM':0,
        ...
    },

}
```

Example of data passed:

```
{
    'params': {'measurement_and_reporting': {'NAME': 'default'}},
    'fixed_labels': [
        {
            'modification': {
                'unimodID': '4',
                'specificity_sites': ['C'],
                'mono_mass': 57.021464,
                'element': {'O': 1, 'H': 3, 'N': 1, 'C': 2},
                'name': 'Carbamidomethyl'
            },
            'AA': 'C'
        }
    ],
    'molecules': 'AA',
    'metabolic_labels': [{'modification': '0, 0.99', 'atom': '15N'}],
    'charges': [1, 2, 3, 4, 5],
    'file': '/BSA1.mzML'
}

Returns:
dict: molecules, evidences, correctly fomatted fixed labels, charges and
    parameters
```

adaptors.**parse_evidence**(*fixed_labels=None*, *evidence_files=None*, *molecules=None*, *evidence_score_field=None*, *return_raw_csv_data=False*)

Reads in the evidence file and returns the final formatted fixed labels, the evidence lookup, which is passed to the isotopologue library and the final formatted molecules (fixed labels are stripped form the molecules).

Note: Output .csv files from Ursgal (Documentation) can directly be used. Also mzTab files can be used as input.

Parameters

- **fixed_labels** (*dict*) – dict with fixed labels, example format is shown below.
- **evidence_files** (*list*) – list of evidence file paths.
- **molecules** (*list*) – list of additional molecules
- **evidence_score_field** (*str*) – specify fieldname which holds the search engine score (Default is "PEP")

Example fixed label format:

```
{
    'C' : [
```

(continues on next page)

(continued from previous page)

```
        {
            'element': {
                'O': 1,
                'H': 3,
                '14N': 1,
                'C': 2
            },
            'evidence_mod_name': 'Carbamidomethyl'
        },
    ]
}
```

Returns final formatted fixed label dict, evidence lookup, list of molecules

Return type tuple

adaptors.**calc_amount_function**(*obj_for_calc_amount*)

Fucntion to calculate actual molecule amounts. Three types of amounts are calculated for a matched isotope chromatogram (MIC), maximum intensity, summed up intensity and area under curve. Additionally the score and the retention time at the maximum intensity are determined.

A test function exists to check correct amount determination.

Returned keys in amound dict:

- 'max I in window'
- 'max I in window (rt)'
- 'max I in window (score)'
- 'auc in window'
- 'sum I in window'

Returns amount dict with keys shown above.

Return type dict

3.6 Parameters

pyQms default params, parsed from current params.py file.

Note: This sphinx source file was **auto-generated** using pyqms/docs/parse_params_for_docu.py, which parses pyqms/params.py Please **do not** modify this file directly, but rather the original parameter files!

```
>>> params = {
        'BUILD_RESULT_INDEX'' : True,
        'COLORS'' : {0.0: (37, 37, 37), 0.1: (99, 99, 99), 0.2: (150, 150, 150), 0.3:␣
↪(204, 204, 204), 0.4: (247, 247, 247), 0.5: (203, 27, 29), 0.6: (248, 120, 72), 0.
↪7: (253, 219, 121), 0.8: (209, 239, 121), 0.9: (129, 202, 78), 1: (27, 137, 62)},
        'ELEMENT_MIN_ABUNDANCE'' : 0.001,
```

(continues on next page)

(continued from previous page)

```
        'FIXED_LABEL_ISOTOPE_ENRICHMENT_LEVELS'' : {'15N': 0.994, '13C': 0.996, '2H':␣
↪0.994},
        'INTENSITY_TRANSFORMATION_FACTOR'' : 100000.0,
        'INTERNAL_PRECISION'' : 1000,
        'LOWER_MZ_LIMIT'' : 150,
        'MACHINE_OFFSET_IN_PPM'' : 0.0,
        'MAX_MOLECULES_PER_MATCH_BIN'' : 20,
        'MINIMUM_NUMBER_OF_MATCHED_ISOTOPOLOGUES'' : 2,
        'MIN_REL_PEAK_INTENSITY_FOR_MATCHING'' : 0.01,
        'MZ_SCORE_PERCENTILE'' : 0.4,
        'MZ_TRANSFORMATION_FACTOR'' : 10000,
        'M_SCORE_THRESHOLD'' : 0.5,
        'PERCENTILE_FORMAT_STRING'' : {0:.3f},
        'REL_I_RANGE'' : 0.2,
        'REL_MZ_RANGE'' : 5e-06,
        'REQUIRED_PERCENTILE_PEAK_OVERLAP'' : 0.5,
        'SILAC_AAS_LOCKED_IN_EXPERIMENT'' : None,
        'UPPER_MZ_LIMIT'' : 2000,
}
```

3.6.1 Descriptions

REQUIRED_PERCENTILE_PEAK_OVERLAP

Defines the percentile how many theoretical and measured peaks must overlap so that the match is considered further. E.g. 0.5 dictates, that 2 of 4 peaks must ovelap

Default value: 0.5

ELEMENT_MIN_ABUNDANCE

Defines the minimum abundance of an element to be considered for the calculation of the isotopologue(s)

Default value: 0.001

MIN_REL_PEAK_INTENSITY_FOR_MATCHING

Defines the relative minimum peak intensity within an isotopologue to be considered for matching

Default value: 0.01

REL_I_RANGE

Defines the relative intensity error range. Represents the relative error to the most intense peak.

Default value: 0.2

REL_MZ_RANGE

Defines the relative m/z error range or the measuring precision of the used mass spectrometer. Is equal to the precision of the used machine in parts per million (ppm)

Default value: 5e-06

MZ_SCORE_PERCENTILE

Defines the weighting between the m/z error and the intensity error for the total score. This weighting can be adjusted for different mass spectrometers, depending on whether m/z or intensity can be measured more accurately

Default value: 0.4

MINIMUM_NUMBER_OF_MATCHED_ISOTOPOLOGUES

Number of isotopologue peaks that are required to yield a mScore. Very small molecules may yield only one isotope peak (monoisotopic peak) or the non-monoisotopic peaks have a very low abundance, so that they ware not considered for macthing

Default value: 2

UPPER_MZ_LIMIT

Defines the maximum m/z value to be considered by pyQms. Can be adjusted for better performance of pyQms or to limit for the measuring range of the used mass spectrometer

Default value: 2000

LOWER_MZ_LIMIT

Defines the minimum m/z value to be considered by pyQms. Can be adjusted for better performance of pyQms or to limit for the measuring range of the used mass spectrometer

Default value: 150

MACHINE_OFFSET_IN_PPM

A mass spectrometer measuring error (constant machine/calibration dependent mass or m/z offset) can be defined here in parts per million (ppm)

Default value: 0.0

M_SCORE_THRESHOLD

The minimum mScore, which should be reported. Typically a mScore above 0.7 yields a FDR below 1%. Lower mScore thresholds can be used to check for machine errors or to optimize matching of pulse-chase samples

Default value: 0.5

SILAC_AAS_LOCKED_IN_EXPERIMENT

These aminoacids have always the defined fixed SILCA modification and their atoms are not considered when calculating a partially labeling percentile

Default value: None

PERCENTILE_FORMAT_STRING

Defines the standard format string when formatting labeling percentile float. Standard format considers three floating points

Default value: {0:.3f}

INTERNAL_PRECISION

Defines the internal precision for float to int conversion

Default value: 1000

MAX_MOLECULES_PER_MATCH_BIN

Defines the number of molecules per match bin. Influences the matching speed

Default value: 20

MZ_TRANSFORMATION_FACTOR

All m/z values are transformed by this factor This value will be multiplied with m/z values before converted to integer. This means that values with a difference of 0.1 ppm @ 1000 m/z won't be distinguishable

Default value: 10000

INTENSITY_TRANSFORMATION_FACTOR

All intensities are transformed with this factor

Default value: 100000.0

BUILD_RESULT_INDEX

The results are indexed for faster access

Default value: True

3.7 Frequently asked questions

3.7.1 Q: What are the hardware requirements for pyQms?

A: pyQms can be run on any (more or less up to date) computer supporting macOS, Linux or Windows and Python version 3.4+. Fast access to spectra is beneficial for the overall performance (e.g. mzML files stored on SSDs). In our experience, slow HDDs (also reading multiple files at the same time from the same HDD or network resource) are most of the time the limiting factor during large scale quantification.

Please consider that the RAM usage dependens on the number of input molecules, charges and labeling percentiles. Some examples are given below.

Molecules #	Charge	label percentiles	RAM [GB]
1000	1-5	None	0.13
10000	1-5	None	0.92
20000	1-5	None	1.76
30000	1-5	None	2.62
10000	1-5	15N 0.0, 0.99	1.90
100	1-5	15N 0.0-0.99, 0.01 steps	1.78

3.7.2 Q: What data/file formats are accepted by pyQms?

A: pyQms accepts simple peak lists consisting of m/z and intensity pairs. E.g.

```
peak_list = [
    ( mz_1, intensity_1 ),
    ( mz_2, intensity_2 ),
    ( mz_n, intensity_n ),
    ...
]
```

Depending on the reader/access to the file format, any input format can be used (mzML, mzXML, RAW, mgf, dta, ...). pyQms comes with pymzML as a dependency, as access to the standard format for mass spectrometry, mzML. It is beneficial, if apart from the peak list also the retention time and the spectrum ID can be provided to pyQms to make data processing and evaluation more straightforward for the user. pyQms comes also with an adaptor to Ursgal (Ursgal) identification csv files, for automated parsing of peptides and modifications.

3.7.3 Q: Does the input data need to be processed?

A: We leave data pre-processing completely on the user side. However, spectra data needs to be centroided. In the example scripts we use pymzML for data centroiding, if the spectra were not already centroided by e.g. Proteome Discoverer or msconvert implemented in Proteowizard.

3.7.4 Q: How should my (input) molecules look like?

A: pyQms accepts different formats of input molecules. Please refer to the documentation of the *pyqms.IsotopologueLibrary* for further details.

Input molecules can be plain peptides (also with modifications in unimod style) or molecular formulas. Please provide multiple molecules in a Python list:

```
'PEPTIDE'
'PEPTIDE+HPO3'
'PEPTIDE#Oxidation:1;Phospho:4'
'+H2O'
```

3.7.5 Q: Is high resolution and low resolution data supported?

A: Since resolved isotope patterns are required, only high resolution data can be processed. The precision can be adjusted in the parameters (*REL_MZ_RANGE*). As default, 5 ppm are used (See: *Parameters*).

3.7.6 Q: Why should I use pyQms to analyze my data?

A: pyQms offers a unique way to quantify all kind of mass spectrometry data including metabolomics, lipidomics and proteomics. All kind of labelings (even completely novel) can be defined and quantified. In contrast to many other algorithms, pyQms will report a score directly reflecting the quanlity of the match, providing the user with useful information and enabling the calculation of FDRs. As a rule-of-thumb, an mScore of 0.7 yields an FDR <= 1% for standard approaches (e.g. label-free or metabolic labeling with 15N). Further, pulse chase data can be analyzed and evaluated. Last but not least, pyQms compares favourably to other popular quantification algorithms in terms of accuracy and sensitivity.

3.7.7 Q: How can i adjust pyQms parameters to my mass spectrometer?

A: Genereally, now extensive adjustements are required. It is normally sufficient to use the default parameters. For further specifications please refer to the *Parameters* section. Most importantly the *REL_MZ_RANGE* has to be set according to the mass spectrometer's accuracy.

3.7.8 Q: Where can I find my final peptide and protein abundances of my LC-MS/MS runs?

A: In pyQms, we offer, on purpose however, no direct estimation of peptide or even protein abundances. We believe, that the user should use the raw quantification data provided by pyQms and determine the abundance with own functions. However, pyQms offers adaptors to read in peptide identification results and use this information to set RT windows and determine e.g. the maximum intensity within this window. Please refer to the *Example Scripts* and *Adaptors* sections for further information and usage examples. We want to keep pyQms open for programmers and tailor the abundance estimation to their needs.

3.7.9 Q: Are there any known issues/problems etc. ?

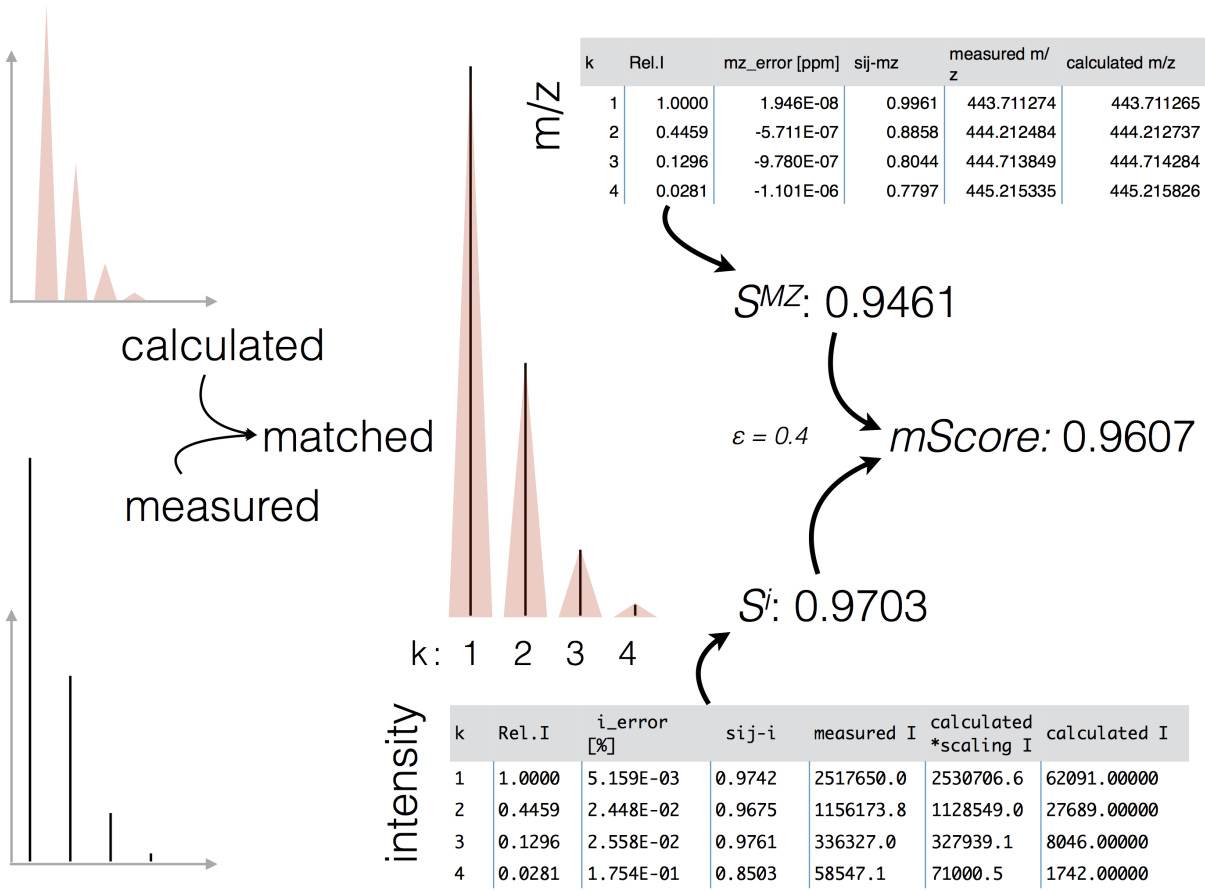
A: So far, no crucial issues or problems were reported. If you encounter any problem feel free to add an issue at GitHub (https://github.com/pyQms/pyqms).

3.7.10 Q: What are the benefits of using pyQms?

A: Besides using a very accurate quantification tool, which is freely available and universally applicable, you and your data will benefit from the concept of the mScore, which adds a new layer of quality assurance to your data analysis.

3.7.11 Q: How does the scoring work? How is the mScore determined?

A: Please refer to the documentation of `pyqms.IsotopologueLibrary` and the publicationfor details on the scoring. The figure below highlights the principle of the mScore and the final score determination originating from the m/z and intensity accuracy scoring.



3.7.12 Q: How can I contribute to the further development of pyQms?

A: Feel free to clone or fork pyQms from GitHub (https://github.com/pyQms/pyqms) and place pull request for your adjustements/improvements/recommendations! Another way is to open an issue at GitHub and let us try to fix it and help you.

3.7.13 Q: I have a problem/issue regarding pyQms, where can I find help?

You can mail us or open an issue at GitHub (https://github.com/pyQms/pyqms) describing your problem/question etc.! We will try to help you.

CHAPTER 4

Examples

4.1 Example Scripts

pyQms comes with mupltiple example script which can be used to test the fucntionality and can be used as templates for own scripts.

4.1.1 Example Scripts

General and quick start

Get the BSA example mzML file

`get_example_BSA_file.main()`

Downloads the BSA.mzML example file also used in openMS and Ursgal.

This file is ideally suited for the use with the example scripts to test pyQms.

Will download the BSA1.mzML to the data folder (../data/BSA1.mzML)

Usage:

./get_example_BSA_file.py

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details
```

(continues on next page)

(continued from previous page)

```
    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import sys
from urllib import request as request
import os
import shutil

def main():
    """
    Downloads the BSA.mzML example file also used in openMS and Ursgal.

    This file is ideally suited for the use with the example scripts to test
    pyQms.

    Will download the BSA1.mzML to the data folder ( ../data/BSA1.mzML )

    Usage:

        ./get_example_BSA_file.py

    """
    mzML_file = os.path.join(
        os.pardir,
        'data',
        'BSA1.mzML'
    )
    if os.path.exists(mzML_file) is False:
        http_url = 'http://sourceforge.net/p/open-ms/code/HEAD/tree/OpenMS/share/
↪OpenMS/examples/BSA/BSA1.mzML?format=raw'
        basename = os.path.basename(http_url).replace('?','') #Win compatible
        output_path = os.path.join( os.path.dirname(mzML_file), basename)
        with open( output_path, 'wb') as ooo:
            local_filename, headers = request.urlretrieve(
                http_url,
                filename = output_path,
            )
        try:
            shutil.move(
                '{0}?format=raw'.format(mzML_file),
                mzML_file
            )
        except:
            shutil.move(
                '{0}format=raw'.format(mzML_file),
                mzML_file
            )
        print(
            'Saved file as {0}'.format(
                mzML_file,
```

(continues on next page)

(continued from previous page)

```
            )
        )

    return


if __name__ == '__main__':
    main()
```

Basic usage

Parse ident file and quantify

parse_ident_file_and_quantify.**main**(*ident_file=None*, *mzml_file=None*)

Script to automatically parse Ursgal result files and quantify it via pyQms. Please refer to Documenation of *Adaptors* for further information.

Ursgal result files or files in *mzTab* format are read in and used for quantification of the BSA example file.

Note: Use e.g. the BSA1.mzML example file. Please download it first using 'get_example_BSA_file.py'. Evidence files can also be found in the data folder 'BSA1_omssa_2_1_9_unified.csv' or 'BSA1_omssa_2_1_9.mztab'

Usage:

./parse_ident_file_and_quantify.py <ident_file> <mzml_file>

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pyqms
import sys
import pickle
import os
import pyqms.adaptors
try:
```

(continues on next page)

(continued from previous page)

```
    import pymzml
except:
    print('Please install pymzML via: pip install pymzml')


def main(ident_file=None, mzml_file=None):
    """
    Script to automatically parse `Ursgal`_ result files and quantify it via
    pyQms. Please refer to Documenation of :doc:`adaptors` for further
    information.

    `Ursgal`_ result files or files in `mzTab` format are read in and used for
    quantification of the BSA example file.

    Note:

        Use e.g. the BSA1.mzML example file. Please download it first using
        'get_example_BSA_file.py'. Evidence files can also be found in the
        data folder 'BSA1_omssa_2_1_9_unified.csv' or 'BSA1_omssa_2_1_9.mztab'

    Usage:

        ./parse_ident_file_and_quantify.py <ident_file> <mzml_file>

    .. _Ursgal:
        https://github.com/ursgal/ursgal

    .. _mzTab:
        http://www.psidev.info/mztab

    """

    if ident_file.upper().endswith('MZTAB'):
        evidence_score_field  = 'search_engine_score[1]'
    else:
        # this is the default value in the adaptor
        evidence_score_field = 'PEP'

    print(
        'Evidence score field "{0}" will be used.'.format(
            evidence_score_field
        )
    )

    fixed_labels, evidences, molecules = pyqms.adaptors.parse_evidence(
        fixed_labels         = None,
        evidence_files       = [ ident_file ],
        evidence_score_field = evidence_score_field
    )

    params = {
        'molecules'        : molecules,
        'charges'          : [1, 2, 3, 4, 5],
        'metabolic_labels' : {'15N' : [0]},
        'fixed_labels'     : fixed_labels,
        'verbose'          : True,
        'evidences'        : evidences
```

(continues on next page)

(continued from previous page)

```
    }

    lib = pyqms.IsotopologueLibrary( **params )

    run = pymzml.run.Reader(
        mzml_file
    )
    out_folder         = os.path.dirname(mzml_file)
    mzml_file_basename = os.path.basename(mzml_file)
    results            = None
    for spectrum in run:
        try:
            # pymzML 2.0.0 style
            scan_time = spectrum.scan_time
        except:
            # scan time will be in seconds
            scan_time = spectrum.get('MS:1000016')
        if spectrum['ms level'] == 1:
            results = lib.match_all(
                mz_i_list = spectrum.centroidedPeaks,
                file_name = mzml_file_basename,
                spec_id   = spectrum['id'],
                spec_rt   = scan_time,
                results   = results
            )
    pickle.dump(
        results,
        open(
            os.path.join(
                out_folder,
                '{0}_pyQms_results.pkl'.format(
                    mzml_file_basename
                )
            ),
            'wb'
        )
    )
    return


if __name__ == '__main__':
    if len( sys.argv ) < 3:
        print(main.__doc__)
    else:
        main(
            ident_file = sys.argv[1],
            mzml_file  = sys.argv[2]
        )
```

Simple match on peak list

basic_quantification_example.**main**(*mzml=None*)

Example script as template for most basic usage of quantification using pyQms.

Use spectrum 1165 of the BSA1.mzML example file. A subrange of the spectrum from m/z 400 to 500 is used.

Usage: ./basic_quantification_example.py

Note: This example does not require a reader to access ms spectra, since a simnple peak liost is used.

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pyqms
import sys
import pickle
import os
import pprint


def main( mzml=None):
    """
    Example script as template for most basic usage of quantification using
    pyQms.

    Use spectrum 1165 of the BSA1.mzML example file. A subrange of the spectrum
    from m/z 400 to 500 is used.

    Usage:
        ./basic_quantification_example.py

    Note:
        This example does not require a reader to access ms spectra, since a
        simnple peak liost is used.

    """

    peak_list = [
        (404.2492407565097, 2652.905029296875),
        (405.3003310237508, 4831.56103515625),
        (408.8403673369115, 23153.7109375),
        (409.17476109421705, 10182.2822265625),
        (409.5098740355617, 4770.97412109375),
        (411.17196124490727, 3454.364013671875),
        (413.26627826402705, 6861.84912109375),
```

(continues on next page)

(continued from previous page)

```
        (419.3157903165357, 90201.5625),
        (420.2440507067882, 11098.4716796875),
        (420.31917273788645, 22288.9140625),
        (420.73825281590496, 8159.7099609375),
        (421.2406187369968, 3768.656494140625),
        (427.3787652898548, 5680.43212890625),
        (433.3316647490907, 8430.30859375),
        (434.705984428002, 25924.38671875),
        (435.2080179219357, 11041.2060546875),
        (443.6708762397708, 4081.282470703125),
        (443.69049198141124, 5107.13330078125),
        (443.6974813419733, 9135.3125),
        (443.7112735313511, 2517650.0),
        (443.7282222289076, 5571.26025390625),
        (443.7379762316008, 5227.4033203125),
        (444.1998579474954, 3021.341796875),
        (444.21248374593875, 1156173.75),
        (444.71384916266277, 336326.96875),
        (445.21533524843596, 58547.0703125),
        (445.71700965093, 4182.04345703125),
        (446.1200302053469, 93216.3359375),
        (447.09963627699824, 3806.537109375),
        (447.1169242266495, 59846.37109375),
        (447.3464079857604, 13170.9541015625),
        (448.11566395552086, 9294.5107421875),
        (448.3500303628631, 3213.052490234375),
        (452.1123280000919, 5092.0869140625),
        (461.1934526664677, 4022.537353515625),
        (462.1463969367603, 99732.5),
        (463.14561508666384, 24247.015625),
        (464.1433022096936, 20417.041015625),
        (465.1421080732791, 3222.4052734375),
        (470.1669593722212, 8621.81640625),
        (475.23989190282134, 3369.073974609375),
        (493.27465300375036, 2725.885986328125),
        (496.0077303201583, 8604.0830078125),
    ]
    print('{0:-^100}'.format('Library generation'))
    lib = pyqms.IsotopologueLibrary(
        molecules        = [ 'DDSPDLPK' ],
        charges          = [ 2 ],
        metabolic_labels = None,
        fixed_labels     = None,
        verbose          = True
    )
    print('{0:-^100}'.format('Library generation'))

    results = lib.match_all(
        mz_i_list = peak_list,
        file_name = 'BSA_test',
        spec_id   = 1165,
        spec_rt   = 29.10,
        results   = None
    )
    print()
    print('{0:-^100}'.format('Results summary'))
    for key in results.keys():
```

(continues on next page)

(continued from previous page)

```
        peptide = results.lookup['formula to molecule'][key.formula][0]
        print(
            'For Peptide {0} with formula {1} and charge {2} the following match␣
↪could be made:'.format(
                peptide,
                key.formula,
                key.charge
            )
        )
        for match in results[key]['data']:
            print(
                '\tAmount {0:1.2f} (scaling_factor) was detected with a matching␣
↪score of {1:1.2f}'.format(
                    match.scaling_factor,
                    match.score
                )
            )
            print(
                '\tThe follwowing peaks have been matched:'
            )
            for measured_mz, measured_intensity, relative_i, calculated_mz,␣
↪calculated_intensity in match.peaks:
                print(
                    '\t\t{0:1.6f} m/z @ {1:1.2e} intensity'.format(
                        measured_mz,
                        measured_intensity
                    )
                )
    print('{0:-^100}'.format('Results summary'))
    return


if __name__ == '__main__':
    main()
```

View result pkl stats

`view_result_pkl_stats.main(result_pkl=None)`

usage: ./view_result_pkl_stats.py <Path2ResultPkl>

This script will show the stats of a result pkl file. Can be used to query the number of quantified formulas and charge states etc.

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:
```

(continues on next page)

(continued from previous page)

```
        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pickle
import sys


def main(result_pkl=None):
    '''

    usage:
        ./view_result_pkl_stats.py <Path2ResultPkl>

    This script will show the stats of a result pkl file. Can be used to query
    the number of quantified formulas and charge states etc.

    '''
    results_class = pickle.load(
        open(
            result_pkl,
            'rb'
        )
    )
    print('Result pkl file holds the following information:')
    print()
    for key, value in results_class.index.items():
        print(
            'Number of {0: <20}: {1}'.format(
                key,
                len(value)
            )
        )
        print('\tExample values (up to 5): {0}'.format(list(value)[:5]))
        print()

if __name__ == '__main__':
    if len( sys.argv ) < 2:
        print(main.__doc__)
    else:
        main(
            result_pkl = sys.argv[1],
        )
```

Access the result class

access_result_class.**main**(*result_pkl=None*)

usage: ./access_result_class.py <Path2ResultPkl>

This script will produce a dictionary with all summed up peptide amounts. The main idea is to show how to

access the result pkl and loop over the data structure.

Note: Since no filters (score, RT windows, etc.) are applied, this script should not be used to estimate the actual amount of the quantified molecules in the results pkl.

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pickle
import sys
import pprint


def main(result_pkl=None):
    '''

    usage:
        ./access_result_class.py <Path2ResultPkl>

    This script will produce a dictionary with all summed up peptide amounts.
    The main idea is to show how to access the result pkl and loop over the
    data structure.

    Note:

        Since no filters (score, RT windows, etc.) are applied, this script
        should not be used to estimate the actual amount of the quantified
        molecules in the results pkl.


    '''
    results_class = pickle.load(
        open(
            result_pkl,
            'rb'
        )
    )

    amount_collector = {}
```

(continues on next page)

(continued from previous page)

```
    for key, value in results_class.items():
        peptide = results_class.lookup['formula to molecule'][key.formula][0]
        if peptide not in amount_collector.keys():
            amount_collector[ peptide ] = {
                'amount' : 0
            }
        for matched_spectrum in value['data']:
            amount_collector[peptide]['amount'] += matched_spectrum.scaling_factor

    pprint.pprint(
        amount_collector
    )


if __name__ == '__main__':
    if len( sys.argv ) < 2:
        print(main.__doc__)
    else:
        main(
            result_pkl = sys.argv[1],
        )
```

Generate quant summary file

generate_quant_summary_file.**main**(*result_pkl=None*)

usage: ./generate_quant_summary_file.py <Path2ResultPkl>

This script will produce quant summary file with all according evidence information which are stored in the result pkl file. Amounts (maxI) will be calculated if possible.

Note: Make sure, an evidence lookup is provided in the results class, so that retention time windows can be defined. Otherwise no meaningful amounts can be calculated.

Warning: Can take very long depending on pkl size!

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
```

(continues on next page)

(continued from previous page)

```
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pickle
import sys


def main(result_pkl=None):
    '''

    usage:
        ./generate_quant_summary_file.py <Path2ResultPkl>

    This script will produce quant summary file with all according evidence
    information which are stored in the result pkl file. Amounts (maxI) will be
    calculated if possible.

    Note:

        Make sure, an evidence lookup is provided in the results class, so that
        retention time windows can be defined. Otherwise no meaningful amounts
        can be calculated.


    Warning:

        Can take very long depending on pkl size!

    '''
    results_class = pickle.load(
        open(
            result_pkl,
            'rb'
        )
    )
    rt_border_tolerance = 1
    # quant_summary_file  = '{0}_quant_summary.csv'.format(result_pkl)
    quant_summary_file  = '{0}_quant_summary.xlsx'.format(result_pkl)
    results_class.write_rt_info_file(
        output_file         = quant_summary_file,
        list_of_csvdicts    = None,
        trivial_name_lookup = None,
        rt_border_tolerance = rt_border_tolerance,
        update              = True
    )
    results_class.calc_amounts_from_rt_info_file(
        rt_info_file         = quant_summary_file,
        rt_border_tolerance  = rt_border_tolerance,
        calc_amount_function = None
    )
    return

if __name__ == '__main__':
    if len( sys.argv ) < 2:
```

(continues on next page)

(continued from previous page)

```
        print(main.__doc__)
    else:
        main(
            result_pkl = sys.argv[1],
        )
```

Write raw quant results as csv

write_raw_result_csv.**main**(*result_pkl=None*)

usage: ./write_raw_result_csv.py <Path2ResultPkl>

Will write all results of a result pkl into a .csv file. Please refer to Documentation of *Result Class* for further information.

Warning: The resulting .csv files can become very large depending on the provided pkl file!

Keys in csv:

- Formula : molecular formula of the molecule (str)
- Molecule : molecule or trivial name (str)
- Charge : charge of the molecule (int)
- ScanID : ScanID of the quantified spectrum (int)
- Label Percentiles : Labeling percentile ((element, enrichment in %),)
- Amount : the determined amount of the molecule
- Retention Time : retetention time of the ScanID
- mScore : score of the isotopologue match
- Filename : filename of spectrum input files

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
```

(continues on next page)

(continued from previous page)

```
import pickle
import sys


def main(result_pkl=None):
    '''

    usage:
        ./write_raw_result_csv.py <Path2ResultPkl>

    Will write all results of a result pkl into a .csv file. Please refer to
    Documentation of :doc:`results` for further information.

    Warning:

        The resulting .csv files can become very large depending on the provided
        pkl file!

    Keys in csv:

        * Formula           : molecular formula of the molecule (str)
        * Molecule          : molecule or trivial name (str)
        * Charge            : charge of the molecule (int)
        * ScanID            : ScanID of the quantified spectrum (int)
        * Label Percentiles : Labeling percentile ( (element, enrichment in %), )
        * Amount            : the determined amount of the molecule
        * Retention Time    : retetention time of the ScanID
        * mScore            : score of the isotopologue match
        * Filename          : filename of spectrum input files

    '''
    results_class = pickle.load(
        open(
            result_pkl,
            'rb'
        )
    )

    results_class.write_result_csv(
        output_file_name= '{0}_raw_results.csv'.format(result_pkl)
    )

if __name__ == '__main__':
    if len( sys.argv ) < 2:
        print(main.__doc__)
    else:
        main(
            result_pkl = sys.argv[1],
        )
```

Write results as mzTab

`write_mztab_result.`**`main`**(*result_pkl=None*)

usage: ./write_mztab_results.py <Path2ResultPkl>

Will write all results of a result pkl into a .mztab file. Please refer to Documentation of *Result Class* for further information.

Note: Please note that the ouput in mzTab format is still in beta stage. Since pyQms is a raw quantification tool, some meta data has to be passed/set manually by the user.

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pickle
import sys


def main(result_pkl=None):
    '''

    usage:
        ./write_mztab_results.py <Path2ResultPkl>

    Will write all results of a result pkl into a .mztab file. Please refer to
    Documentation of :doc:`results` for further information.

    Note:

        Please note that the ouput in mzTab format is still in beta stage.
        Since pyQms is a raw quantification tool, some meta data has to be
        passed/set manually by the user.

    '''
    results_class = pickle.load(
        open(
            result_pkl,
            'rb'
        )
    )

    results_class.write_result_mztab(
        output_file_name = '{0}_results.mztab'.format(result_pkl)
    )
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    if len( sys.argv ) < 2:
        print(main.__doc__)
    else:
        main(
            result_pkl = sys.argv[1],
        )
```

Write results as mzTab (BSA example)

write_BSA_mztab_results.**main**(*result_pkl=None*)

usage: ./write_mztab_results.py <Path2ResultPkl>

Will write all results of a result pkl into a .mztab file. Please refer to Documentation of *Result Class* for further information.

Warning: This example script is specifically for the BSA1.mzML quantification results, since file specific meta data is passed. Please use 'write_mztab_results.py' for a more general script to produce mzTab results.

Note: Please note that the ouput in mzTab format is still in beta stage. Since pyQms is a raw quantification tool, some meta data has to be passed/set manually by the user.

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pickle
import sys


def main(result_pkl=None):
    '''
```

(continues on next page)

(continued from previous page)

```
    usage:
        ./write_mztab_results.py <Path2ResultPkl>

    Will write all results of a result pkl into a .mztab file. Please refer to
    Documentation of :doc:`results` for further information.

    Warning:

        This example script is specifically for the BSA1.mzML quantification
        results, since file specific meta data is passed. Please use
        'write_mztab_results.py' for a more general script to produce mzTab
        results.

    Note:

        Please note that the ouput in mzTab format is still in beta stage.
        Since pyQms is a raw quantification tool, some meta data has to be
        passed/set manually by the user.

    '''
    results_class = pickle.load(
        open(
            result_pkl,
            'rb'
        )
    )
    # provide meta data as lists of mztab specific formats. Pass directly
    # mztab correct format.

    mztab_meta_info = {
        'protein_search_engine_score'   : [],
        'psm_search_engine_score'       : ['[MS,MS:1001475,OMSSA:evalue, ]'],
        'fixed_mod'                     : ['[UNIMOD, UNIMOD:4, Carbamidomethyl, ]'],
        'variable_mod'                  : ['[UNIMOD, UNIMOD:35, Oxidation, ]'],
        'study_variable-description'    : ['Standard BSA measurement'],
        'ms_run-location'               : ['BSA1.mzML'],

    }

    results_class.lookup['mztab_meta_info'] = mztab_meta_info

    results_class.write_result_mztab(
        output_file_name = '{0}_results.mztab'.format(result_pkl)
    )

if __name__ == '__main__':
    if len( sys.argv ) < 2:
        print(main.__doc__)
    else:
        main(
            result_pkl = sys.argv[1],
        )
```

Advanced usage

Parse ident file and quantify (with CAM)

parse_ident_file_and_quantify_with_carbamidomethylation.**main**(*ident_file=None*, *mzml_file=None*)

Script to automatically parse Ursgal result files and quantify it via pyQms.

For evidence files with molecules with Caramidomethylation as fixed modification. These mode will be stripped from the molecules. This is important if an metabolic label (like 15N) is applied. This ensures that the nitrogens pools of the peptides (which are 15N labeled) do not mix up with the nitrogen pool of the Carbamidomethylation (14N since intriduced during sample preparation). Please refer to Documenation of *Adaptors* for further information.

Ursgal result files or files in *mzTab* format are read in and used for quantification of the BSA example file.

Note: Use e.g. the BSA1.mzML example file. Please download it first using 'get_example_BSA_file.py'. Evidence files can also be found in the data folder 'BSA1_omssa_2_1_9_unified.csv' or 'BSA1_omssa_2_1_9.mztab'

Usage:

./parse_ident_file_and_quantify_with_carbamidomethylation.py <ident_file> <mzml_file>

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pyqms
import sys
import pickle
import os
import pyqms.adaptors
try:
    import pymzml
except:
    print('Please install pymzML via: pip install pymzml')


def main(ident_file=None, mzml_file=None):
    '''

    Script to automatically parse `Ursgal`_ result files and quantify it via
```

(continues on next page)

(continued from previous page)

```
    pyQms.

    For evidence files with molecules with Caramidomethylation as fixed
    modification. These mode will be stripped from the molecules. This is
    important if an metabolic label (like 15N) is applied. This ensures that the
    nitrogens pools of the peptides (which are 15N labeled) do not mix up with
    the nitrogen pool of the Carbamidomethylation (14N since intriduced during
    sample preparation). Please refer to Documenation of :doc:`adaptors` for
    further information.

    `Ursgal`_ result files or files in `mzTab` format are read in and used for
    quantification of the BSA example file.

    Note:

        Use e.g. the BSA1.mzML example file. Please download it first using
        'get_example_BSA_file.py'. Evidence files can also be found in the
        data folder 'BSA1_omssa_2_1_9_unified.csv' or 'BSA1_omssa_2_1_9.mztab'

    Usage:

        ./parse_ident_file_and_quantify_with_carbamidomethylation.py <ident_file>
↪<mzml_file>

    .. _Ursgal:
        https://github.com/ursgal/ursgal

    .. _mzTab:
        http://www.psidev.info/mztab

    '''

    # define the fixed label for Caramidomethyl
    tmp_fixed_labels = {
        'C' : [
            {
                'element_composition' : {'O': 1, 'H': 3, '14N': 1, 'C': 2},
                'evidence_mod_name': 'Carbamidomethyl'
            },
        ]
    }

    formatted_fixed_labels, evidence_lookup, molecule_list = pyqms.adaptors.parse_
↪evidence(
        fixed_labels   = tmp_fixed_labels,
        evidence_files = [ ident_file ],
    )

    params = {
        'molecules'        : molecule_list,
        'charges'          : [1, 2, 3, 4, 5],
        'metabolic_labels' : {'15N' : [0, ]},
        'fixed_labels'     : formatted_fixed_labels,
        'verbose'          : True,
        'evidences'        : evidence_lookup
    }
```

(continues on next page)

(continued from previous page)

```
    lib = pyqms.IsotopologueLibrary( **params )

    run = pymzml.run.Reader(
        mzml_file
    )
    out_folder         = os.path.dirname(mzml_file)
    mzml_file_basename = os.path.basename(mzml_file)
    results = None
    for spectrum in run:
        spec_id = spectrum['id']
        try:
            # pymzML 2.0.0 style
            scan_time = spectrum.scan_time
        except:
            # scan time will be in seconds
            scan_time = spectrum.get('MS:1000016')
        if spectrum['ms level'] == 1:
            results = lib.match_all(
                mz_i_list = spectrum.centroidedPeaks,
                file_name = mzml_file_basename,
                spec_id   = spectrum['id'],
                spec_rt   = scan_time,
                results   = results
            )

    pickle.dump(
        results,
        open(
            os.path.join(
                out_folder,
                '{0}_pyQms_results.pkl'.format(
                    mzml_file_basename
                )
            ),
            'wb'
        )
    )
    return


if __name__ == '__main__':
    if len( sys.argv ) < 3:
        print(main.__doc__)
    else:
        main(
            ident_file = sys.argv[1],
            mzml_file  = sys.argv[2]
        )
```

Complete quantification - from identification csv to peptide abundances

complete_BSA_quantification.**main**(*ident_file=None*, *mzml_file=None*)

Examples script to demonstrate a (example) workflow from mzML files to peptide abundances. Will plot for every quantified peptide a matched isotopologue chromatogram (MIC). The plots include RT windows, maximum amount in RT window and identification RT(s).

Ursgal result files or files in *mzTab* format are read in and used for quantification of the BSA example file.

Note: Use e.g. the BSA1.mzML example file. Please download it first using 'get_example_BSA_file.py'. Evidence files can also be found in the data folder 'BSA1_omssa_2_1_9_unified.csv' or 'BSA1_omssa_2_1_9.mztab'

Usage:

./complete_BSA_quantification.py <ident_file> <mzml_file>

Note: rpy2 is required for all plotting

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pyqms
import sys

import pickle
import os
import pyqms.adaptors
try:
    import pymzml
except:
    print('Please install pymzML via: pip install pymzml')


def main(ident_file = None, mzml_file = None):
    '''
    Examples script to demonstrate a (example) workflow from mzML files to
    peptide abundances. Will plot for every quantified peptide a matched
    isotopologue chromatogram (MIC). The plots include RT windows, maximum
    amount in RT window and identification RT(s).

    `Ursgal`_ result files or files in `mzTab` format are read in and used for
    quantification of the BSA example file.

    Note:
```

(continues on next page)

(continued from previous page)

```
        Use e.g. the BSA1.mzML example file. Please download it first using
        'get_example_BSA_file.py'. Evidence files can also be found in the
        data folder 'BSA1_omssa_2_1_9_unified.csv' or 'BSA1_omssa_2_1_9.mztab'

    Usage:

        ./complete_BSA_quantification.py <ident_file> <mzml_file>

    .. _Ursgal:
        https://github.com/ursgal/ursgal

    .. _mzTab:
        http://www.psidev.info/mztab

    Note:
        rpy2 is required for all plotting

    '''

    # define the fixed label for Carbamidomethyl
    tmp_fixed_labels = {
        'C' : [
            {
                'element_composition' : {'O': 1, 'H': 3, '14N': 1, 'C': 2},
                'evidence_mod_name': 'Carbamidomethyl'
            },
        ]
    }
    if ident_file.upper().endswith('MZTAB'):
        evidence_score_field  = 'search_engine_score[1]'
    else:
        # this is the default value in the adaptor
        evidence_score_field = 'PEP'

    print(
        'Evidence score field "{0}" will be used.'.format(
            evidence_score_field
        )
    )
    formatted_fixed_labels, evidence_lookup, molecule_list = pyqms.adaptors.parse_
↪evidence(
        fixed_labels         = tmp_fixed_labels,
        evidence_files       = [ ident_file ],
        evidence_score_field = evidence_score_field
    )

    params = {
        'molecules'        : molecule_list,
        'charges'          : [1, 2, 3, 4, 5],
        'metabolic_labels' : {'15N' : [0, ]},
        'fixed_labels'     : formatted_fixed_labels,
        'verbose'          : True,
        'evidences'        : evidence_lookup
    }

    lib = pyqms.IsotopologueLibrary( **params )
```

(continues on next page)

(continued from previous page)

```
    run = pymzml.run.Reader(
        mzml_file
    )
    out_folder         = os.path.dirname(mzml_file)
    mzml_file_basename = os.path.basename(mzml_file)
    results = None
    for spectrum in run:
        spec_id   = spectrum['id']
        try:
            # pymzML 2.0.0 style
            scan_time = spectrum.scan_time
        except:
            # scan time will be in seconds
            scan_time = spectrum.get('MS:1000016')
        if spectrum['ms level'] == 1:
            results = lib.match_all(
                mz_i_list = spectrum.centroidedPeaks,
                file_name = mzml_file_basename,
                spec_id   = spectrum['id'],
                spec_rt   = scan_time,
                results   = results
            )
    # print(results)
    out_folder = os.path.join(
        os.path.dirname(ident_file),
        'complete_BSA_quantification'
    )
    if os.path.exists(out_folder) is False:
        os.mkdir(out_folder)
    print()
    print('All results go into folder: {0}'.format(out_folder))
    rt_border_tolerance = 1
    quant_summary_file  = os.path.join(
        out_folder,
        'complete_BSA_quantification_summary.xlsx',
    )
    results.write_rt_info_file(
        output_file         = quant_summary_file,
        list_of_csvdicts    = None,
        trivial_name_lookup = None,
        rt_border_tolerance = rt_border_tolerance,
        update              = True
    )
    calculated_amounts = results.calc_amounts_from_rt_info_file(
        rt_info_file         = quant_summary_file,
        rt_border_tolerance  = rt_border_tolerance,
        calc_amount_function = None, # calc_amount_function
    )
    # print(calculated_amounts)
    formula_charge_to_quant_info = {}
    for line_dict in calculated_amounts:
        formula_charge_to_quant_info[ (line_dict['formula'], int(line_dict['charge
↪'])) ] = {
            'rt'           : line_dict['max I in window (rt)'],
            'amount'       : line_dict['max I in window'],
            'rt start'     : line_dict['start (min)'],
```

(continues on next page)

(continued from previous page)

```
            'rt stop'      : line_dict['stop (min)'],
            'evidence_rts' : [],
        }
        if len(formula_charge_to_quant_info[ (line_dict['formula'], int(line_dict[
↪'charge'])) ]['evidence_rts']) == 0:
            for ev_string in line_dict['evidences (min)'].split(';'):
                formula_charge_to_quant_info[ (line_dict['formula'], int(line_dict[
↪'charge'])) ]['evidence_rts'].append(
                    round( float( ev_string.split('@')[1] ), 2 )
                )
    import_ok = False
    try:
        import rpy2
        import_ok = True
    except:
        pass
    if import_ok:
        print('Plotting results plot including RT windows, abundances and␣
↪identifications')
        for key in results.keys():
            short_key = ( key.formula, key.charge )

            match_list = results[key]['data']
            if len(match_list) < 15:
                continue
            file_name = os.path.join(
                out_folder ,
                'MIC_2D_{0}_{1}.pdf'.format(
                    '_'.join(
                        results.lookup['formula to molecule'][ key.formula ]
                    ),
                    key.charge,
                )
            )
            graphics, grdevices = results.init_r_plot(file_name)

            ablines = {
                key : [
                    {
                        'v'   : formula_charge_to_quant_info[short_key]['rt'],
                        'lty' : 2
                    },
                    {
                        'v'   : formula_charge_to_quant_info[short_key]['rt start'],
                        'lty' : 2,
                        'col' : 'blue'
                    },
                    {
                        'v'   : formula_charge_to_quant_info[short_key]['rt stop'],
                        'lty' : 2,
                        'col' : 'blue'
                    },
                ]
            }
            # print(formula_charge_to_quant_info[short_key])
            additional_legends = {
                key : [
```

(continues on next page)

(continued from previous page)

```
                    {
                        'x'    : formula_charge_to_quant_info[short_key]['rt'],
                        'y'    : formula_charge_to_quant_info[short_key]['amount'],
                        'text' : 'max intensity: {0:1.3e}'.format(
                            formula_charge_to_quant_info[short_key]['amount'],
                        ),
                        'pos'  : 3 # above
                    },
                    {
                        'x'    : formula_charge_to_quant_info[short_key]['rt start'],
                        'y'    : formula_charge_to_quant_info[short_key]['amount'] /␣
↪2,
                        'text' : 'RT Window start',
                        'pos'  : 4, # right
                        'col'  : 'blue'
                    },
                    {
                        'x'    : formula_charge_to_quant_info[short_key]['rt stop'],
                        'y'    : formula_charge_to_quant_info[short_key]['amount'] /␣
↪2,
                        'text' : 'RT window stop',
                        'pos'  : 2, # left,
                        'col'  : 'blue'
                    },
                ]
            }

            for evidence_rt in formula_charge_to_quant_info[short_key]['evidence_rts
↪']:
                ablines[key].append(
                    {
                        'v'   : evidence_rt,
                        'lwd' : 0.5,
                        'col' : 'purple',
                    }
                )
                additional_legends[key].append(
                    {
                        'x'    : evidence_rt,
                        'y'    : 0,
                        'lwd'  : 0.5,
                        'col'  : 'purple',
                        'text' : 'MS2 ident',
                        'pos'  : 4,
                        'srt'  : 45 # rotate label
                    }
                )

            results.plot_MICs_2D(
                [key],
                file_name          = None,
                rt_window          = None,
                i_transform        = None,
                xlimits            = [
                    formula_charge_to_quant_info[short_key]['rt start']-0.05,
                    formula_charge_to_quant_info[short_key]['rt stop']+0.05,
```

(continues on next page)

(continued from previous page)

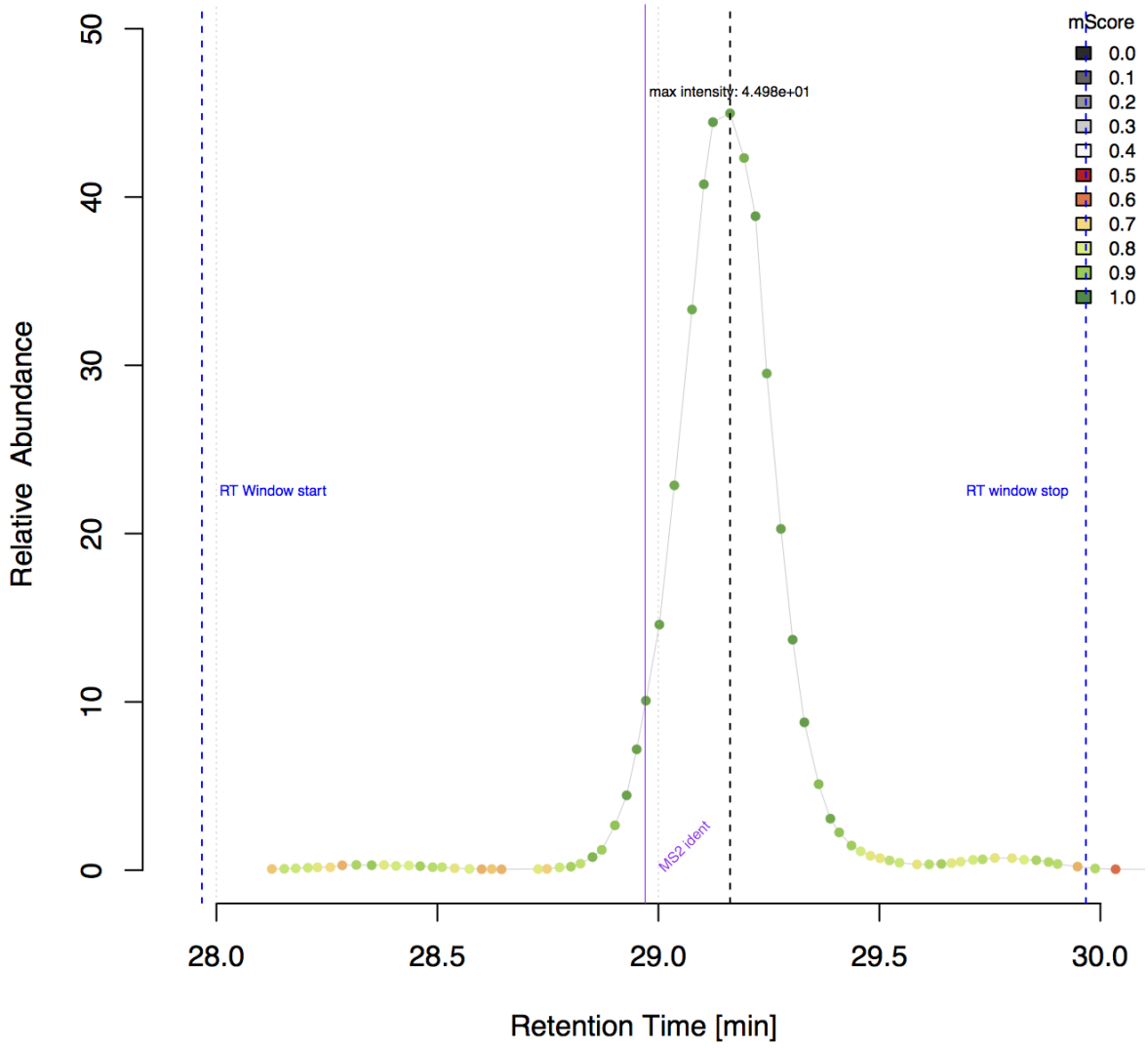
```
                ],
                additional_legends = additional_legends,
                title              = None,
                zlimits            = None,
                ablines            = ablines,
                graphics           = graphics
            )
            print(
                'Plottted {0}'.format(file_name)
            )

    return


if __name__ == '__main__':
    if len( sys.argv ) < 3:
        print(main.__doc__)
    else:
        main(
            ident_file = sys.argv[1],
            mzml_file  = sys.argv[2]
        )
```

Example plot including RT borders and identification information

Example plot for peptide 'DDSPDLPK' with charge 2 in the BSA1.mzML file.



Plotting and visualizattion

Plot example match

`plot_match_examples.`**`main`**(*result_pkl=None*)

usage: ./plot_match_examples.py <Path2ResultPkl>

Extracts the match information and plots one example isotopologue match into the 'data' folder. Uses the plot function of pymzML (pymzML.plot). Use this script as template for annotating spectra with match information.

Note: Plots only one high scored formula (mScore >0.95) from the result pkl. Use e.g. with the 'BSA1.mzML_pyQms_results.pkl' obtained from e.g. example script

parse_ident_file_and_quantify_with_carbamidomethylation.py to get example plotting data.

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pickle
import sys
import os
try:
    import pymzml
    import pymzml.plot
except:
    print('Please install pymzML via: pip install pymzml')


def main(result_pkl=None):
    '''

    usage:
        ./plot_match_examples.py <Path2ResultPkl>

    Extracts the match information and plots one example isotopologue match into
    the 'data' folder. Uses the plot function of pymzML (`pymzML.plot`_). Use
    this script as template for annotating spectra with match information.

    Note:

        Plots only one high scored formula (mScore >0.95) from the result pkl.
        Use e.g. with the 'BSA1.mzML_pyQms_results.pkl' obtained from e.g.
        example script parse_ident_file_and_quantify_with_carbamidomethylation.py
        to get example plotting data.

    .. _pymzML.plot:
        https://pymzml.github.io/plot.html

    '''
    results_class = pickle.load(
        open(
            result_pkl,
            'rb'
```

(continues on next page)

(continued from previous page)

```
        )
    )

    for key, i, entry in results_class.extract_results():
        if entry.score > 0.95:
            p = pymzml.plot.Factory()
            label_x           = []
            measured_peaks    = []
            matched_peaks     = []
            for measured_mz, measured_intensity, relative_i, calculated_mz,␣
↪calculated_intensity in entry.peaks:
                if measured_mz is not None:
                    measured_peaks.append( (measured_mz, measured_intensity) )
                    matched_peaks.append( (calculated_mz, calculated_intensity *␣
↪entry.scaling_factor) )
                    label_x.append(
                        (
                            calculated_mz,
                            '{0:5.3f} ppm'.format(
                                (measured_mz - calculated_mz) / ( measured_mz * 1e-6 )
                            )
                        )
                    )

            mz_only  = [ n[0] for n in measured_peaks ]
            mz_range = [ min(mz_only)-1, max(mz_only)+1 ]
            peptides = results_class.lookup['formula to molecule'][key.formula]
            if len(peptides) > 1:
                continue
            p.newPlot(
                header = 'Formula: {0}; Peptide: {1}; Charge: {2}\n File: {3}; Scan:
↪{4}; RT: {5:1.3f}\n Amount: {6:1.3f}; Score: {7:1.3f}'.format(
                    key.formula,
                    peptides[0],
                    key.charge,
                    key.file_name,
                    entry.spec_id,
                    entry.rt,
                    entry.scaling_factor,
                    entry.score
                ),
                mzRange = mz_range
            )
            p.add(
                measured_peaks,
                color = (0, 0, 0),
                style = 'sticks'
            )
            p.add(
                matched_peaks,
                color = (0, 200, 0),
                style = 'triangles'
            )
            p.add(
                label_x,
                color = (0, 0, 255),
                style = 'label_x'
```

(continues on next page)

(continued from previous page)

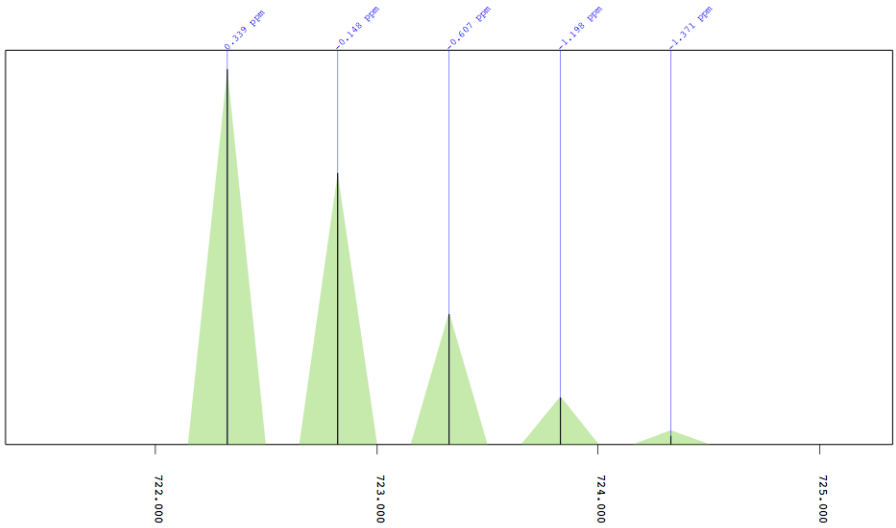
```
            )

            plot_name = os.path.join(
                os.pardir,
                'data',
                '{0}_Peptide_{1}_Charge_{2}.xhtml'.format(
                    key.file_name,
                    peptides[0],
                    key.charge
                )
            )
            p.save(
                filename = plot_name,
                mzRange  = mz_range
            )
            print(
                'Plotted file {0}'.format(
                    plot_name
                )
            )
            break


if __name__ == '__main__':
    if len( sys.argv ) < 2:
        print(main.__doc__)
    else:
        main(
            result_pkl = sys.argv[1],
        )
```

Example match plot

```
Formula: C(59)H(94)14N(1)N(15)O(24)S(1); Peptide: YIC0DNQDTISSK; Charge: 2
File: BSA1.mzML; Scan: 1193; RT: 29.855
Amount: 31.989; Score: 0.960
```



MIC 3D plot

mic_3d_plot.**main**(*pickle_file*)

usage: ./mic_3d_plot.py <path_to_pickled_result_class>

Plots 3-dimensional matched isotope chromatograms (MICs) of pyQms quantification results.

Pickled result class can contain thousands of molecules therefore this example script stops plotting after 10 plotted MICs. Otherwise all quantified formula-charge-filename combinations will be plotted!

Use e.g. the BSA data example. Download via 'get_example_BSA_file.py' and quantify using 'parse_ident_file_and_quantify_with_carbmidomethylation.py'.

Note: Installation of R and rpy2 is required.

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details
```

(continues on next page)

(continued from previous page)

```
    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""

import pickle
import sys
import os

try:
    import rpy2
except:
    print('rpy2 is not installed but required for plotting, please install it and try␣
↪again')
    print('pip3.4 install rpy2')


def main(pickle_file):
    '''
    usage:
        ./mic_3d_plot.py <path_to_pickled_result_class>

    Plots 3-dimensional matched isotope chromatograms (MICs) of pyQms
    quantification results.

    Pickled result class can contain thousands of molecules therefore this
    example script stops plotting after 10 plotted MICs. Otherwise all
    quantified formula-charge-filename combinations will be plotted!

    Use e.g. the BSA data example. Download via 'get_example_BSA_file.py' and
    quantify using 'parse_ident_file_and_quantify_with_carbmidomethylation.py'.

    Note:

        Installation of R and rpy2 is required.

    '''
    results = pickle.load(
        open( pickle_file, 'rb')
    )
    out_folder = os.path.join(
        os.path.dirname(pickle_file),
        'plots'
    )
    if os.path.exists(out_folder) is False:
        os.mkdir(out_folder)
    print('Plotting into folder: {0}'.format(out_folder))
    if len( results.keys() ) > 10:
        print(
            '''
```

(continues on next page)

(continued from previous page)

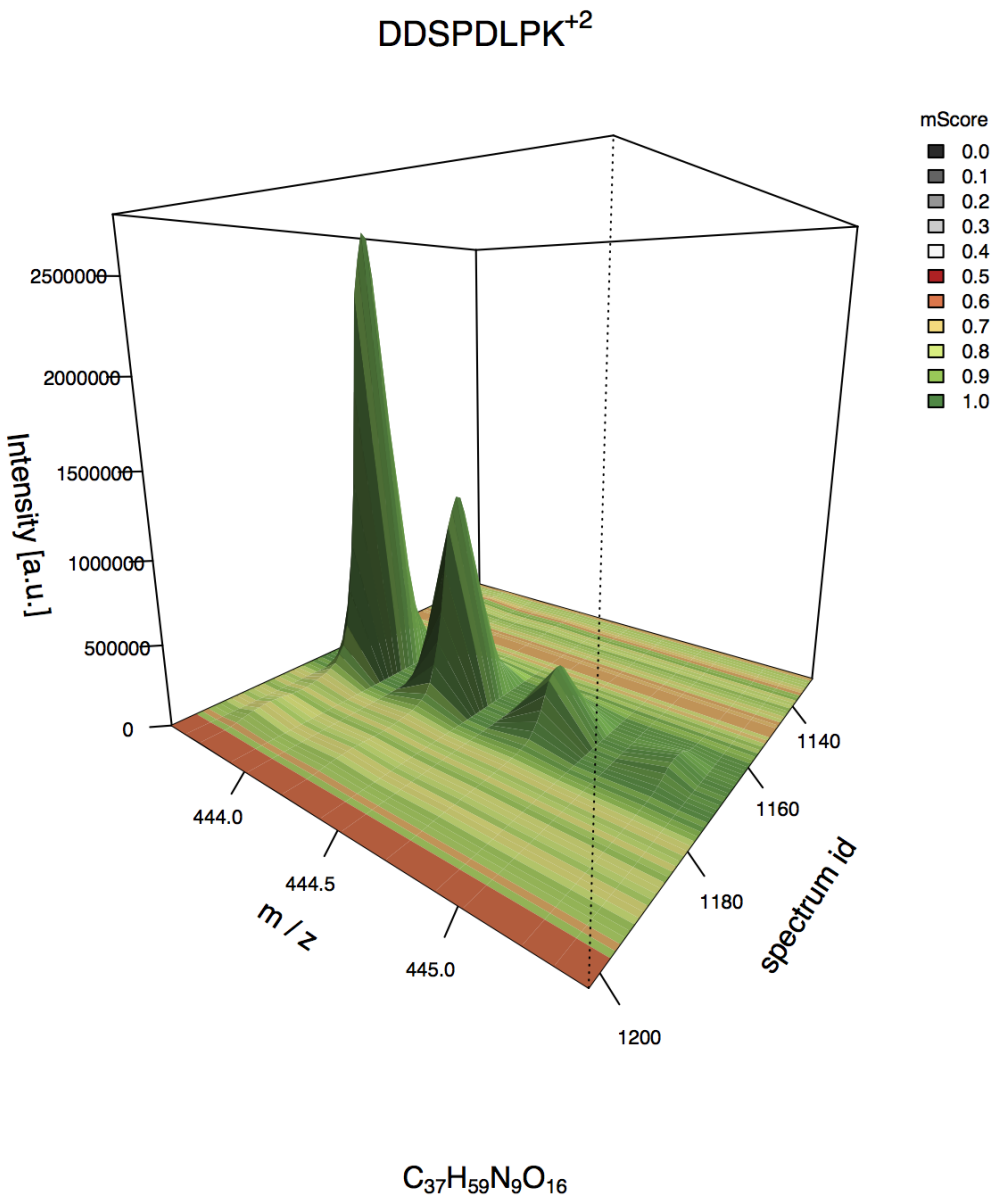
```
Result class should not hold more then 10 keys, to prevent plot overflow!
Will stop after 10 plots!
'''
        )
        # sys.exit()
    for n, key in enumerate(results.keys()):
        if n > 10:
            print('Stopping after 10 plots!')
            exit()
        if len(results[key]['data']) <= 15:
            continue
        mzml_filename = key.file_name
        if os.sep in mzml_filename:
            mzml_filename = os.path.basename(mzml_filename)

        file_name = os.path.join(
            out_folder ,
            'MIC_3D_{0}_{1}_{2}_{3}'.format(
                '_'.join(
                    results.lookup['formula to molecule'][ key.formula ]
                ),
                key.charge,
                key.label_percentiles,
                mzml_filename
            )
        )
        results.plot_MIC_3D(
            key,
            file_name = file_name,
        )

    return


if __name__ == '__main__':
    if len(sys.argv) <= 1:
        sys.exit(main.__doc__)
    main( sys.argv[1] )
```

Example 3D plot



MIC 2D plot

`mic_2d_plot.main`(*pickle_file*)

usage: ./mic_2d_plot.py <path_to_pickled_result_class>

Plots 2-dimensional matched isotope chromatograms (MICs) of pyQms quantification results.

Pickled result class can contain thousands of molecules therefore this example script stops plotting after 10 plotted MICs. Otherwise all quantified formula-charge-filename combinations will be plotted!

Use e.g. the BSA data example. Download via 'get_example_BSA_file.py' and quantify using

'parse_ident_file_and_quantify_with_carbmidomethylation.py'.

Note: Installation of R and rpy2 is required.

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""

import pickle
import sys
import os

try:
    import rpy2
except:
    print('rpy2 is not installed but required for plotting, please install it and try␣
↪again')
    print('pip3.4 install rpy2')


def main(pickle_file):
    '''
    usage:
        ./mic_2d_plot.py <path_to_pickled_result_class>

    Plots 2-dimensional matched isotope chromatograms (MICs) of pyQms
    quantification results.

    Pickled result class can contain thousands of molecules therefore this
    example script stops plotting after 10 plotted MICs. Otherwise all
    quantified formula-charge-filename combinations will be plotted!

    Use e.g. the BSA data example. Download via 'get_example_BSA_file.py' and
    quantify using 'parse_ident_file_and_quantify_with_carbmidomethylation.py'.

    Note:

        Installation of R and rpy2 is required.
```

(continues on next page)

(continued from previous page)

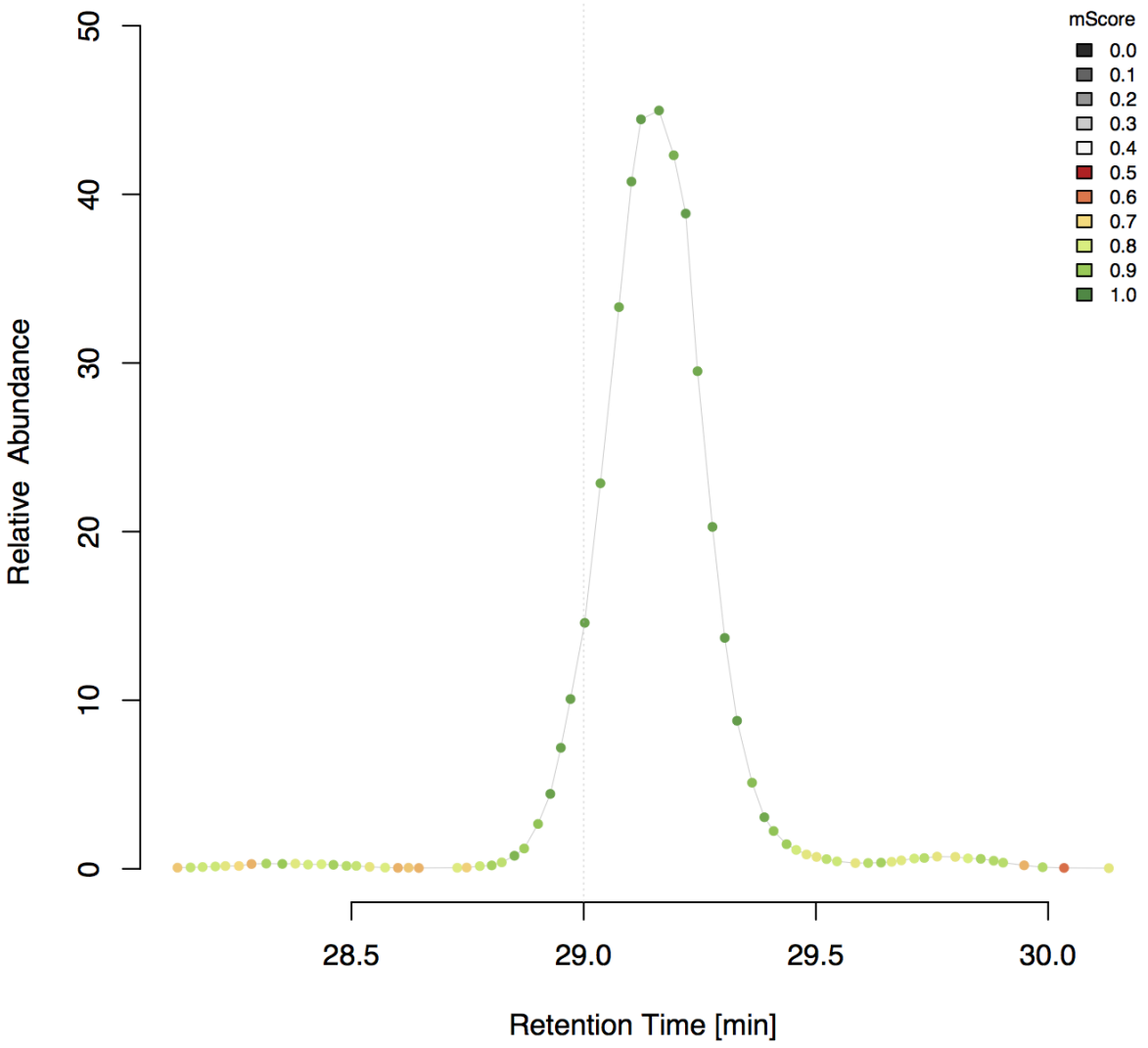
```
    '''
    results = pickle.load(
        open( pickle_file, 'rb')
    )
    out_folder = os.path.join(
        os.path.dirname(pickle_file),
        'plots'
    )
    if os.path.exists(out_folder) is False:
        os.mkdir(out_folder)
    print('Plotting into folder: {0}'.format(out_folder))
    if len( results.keys() ) > 10:
        print(
            '''
Result class should not hold more then 10 keys, to prevent plot overflow!
Will stop after 10 plots!
'''
        )
        # sys.exit()
    for n, key in enumerate(results.keys()):
        if n > 10:
            print('Stopping after 10 plots!')
            exit()
        if len(results[key]['data']) <= 15:
            continue
        mzml_filename = key.file_name
        if os.sep in mzml_filename:
            mzml_filename = os.path.basename(mzml_filename)

        file_name = os.path.join(
            out_folder ,
            'MIC_2D_{0}_{1}_{2}_{3}.pdf'.format(
                '_'.join(
                    results.lookup['formula to molecule'][ key.formula ]
                ),
                key.charge,
                key.label_percentiles,
                mzml_filename
            )
        )
        graphics, grdevices = results.init_r_plot(file_name)
        results.plot_MICs_2D(
            [key],
            graphics = graphics
        )


    return


if __name__ == '__main__':
    if len(sys.argv) <= 1:
        sys.exit(main.__doc__)
    main( sys.argv[1] )
```

Example 2D plot



Determine m/z and intensity errors

determine_mz_and_i_error.**main**(*result_pkl=None*)

usage: ./determine_mz_and_i_error.py <Path2ResultPkl>

This script will determine the apparant mz and intensity error present in the quantifications for the given result pkl.

```
#!/usr/bin/env python3
# encoding: utf-8
```

(continues on next page)

(continued from previous page)

```
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pickle
import sys
import os


def main(result_pkl=None):
    '''

    usage:
        ./determine_mz_and_i_error.py <Path2ResultPkl>

    This script will determine the apparant mz and intensity error present
    in the quantifications for the given result pkl.


    '''
    results_class = pickle.load(
        open(
            result_pkl,
            'rb'
        )
    )

    plot_name = os.path.join(
        os.path.dirname(result_pkl),
        'mz_and_intensity_error_{0}.pdf'.format(
            os.path.basename(result_pkl)
        )
    )

    results_class._determine_measured_error(
        score_threshold = None,
        topX            = 3,
        filename        = plot_name,
        plot            = True
    )


if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    if len( sys.argv ) < 2:
        print(main.__doc__)
    else:
        main(
            result_pkl = sys.argv[1],
        )
```

Visualize errors on spectrum level

visualize_scoring_information.**main**(*mzml=None*)

Example script fort visualizing the m/z and intensity error, which is the basis for the scoring of the matches in pyQms.

Use spectrum 1165 of the BSA1.mzML example file. A subrange of the spectrum from m/z 400 to 500 is used.

Usage: ./visualize_scoring_information.py

Note: This example does not require a reader to access MS spectra, since a simnple peak list is used.

```
#!/usr/bin/env python3
# encoding: utf-8
"""
    pyQms
    -----

    Python module for fast and accurate mass spectrometry data quantification

    :license: MIT, see LICENSE.txt for more details

    Authors:

        * Leufken, J.
        * Niehues, A.
        * Sarin, L.P.
        * Hippler, M.
        * Leidel, S.A.
        * Fufezan, C.

"""
import pyqms
import sys
import pickle
import os
import pprint
from collections import defaultdict as ddict
try:
    import pymzml
except:
    print('Please install pymzML via: pip install pymzml')


def main( mzml=None):
    """
```

(continues on next page)

(continued from previous page)

```
    Example script fort visualizing the m/z and intensity error, which is the
    basis for the scoring of the matches in pyQms.

    Use spectrum 1165 of the BSA1.mzML example file. A subrange of the spectrum
    from m/z 400 to 500 is used.

    Usage:
        ./visualize_scoring_information.py

    Note:
        This example does not require a reader to access MS spectra, since a
        simnple peak list is used.

    """

    peak_list = [
        (404.2492407565097, 2652.905029296875),
        (405.3003310237508, 4831.56103515625),
        (408.8403673369115, 23153.7109375),
        (409.17476109421705, 10182.2822265625),
        (409.5098740355617, 4770.97412109375),
        (411.17196124490727, 3454.364013671875),
        (413.26627826402705, 6861.84912109375),
        (419.3157903165357, 90201.5625),
        (420.2440507067882, 11098.4716796875),
        (420.31917273788645, 22288.9140625),
        (420.73825281590496, 8159.7099609375),
        (421.2406187369968, 3768.656494140625),
        (427.3787652898548, 5680.43212890625),
        (433.3316647490907, 8430.30859375),
        (434.705984428002, 25924.38671875),
        (435.2080179219357, 11041.2060546875),
        (443.6708762397708, 4081.282470703125),
        (443.69049198141124, 5107.13330078125),
        (443.6974813419733, 9135.3125),
        (443.7112735313511, 2517650.0),
        (443.7282222289076, 5571.26025390625),
        (443.7379762316008, 5227.4033203125),
        (444.1998579474954, 3021.341796875),
        (444.21248374593875, 1156173.75),
        (444.71384916266277, 336326.96875),
        (445.21533524843596, 58547.0703125),
        (445.71700965093, 4182.04345703125),
        (446.1200302053469, 93216.3359375),
        (447.09963627699824, 3806.537109375),
        (447.1169242266495, 59846.37109375),
        (447.3464079857604, 13170.9541015625),
        (448.11566395552086, 9294.5107421875),
        (448.3500303628631, 3213.052490234375),
        (452.1123280000919, 5092.0869140625),
        (461.1934526664677, 4022.537353515625),
        (462.1463969367603, 99732.5),
        (463.14561508666384, 24247.015625),
        (464.1433022096936, 20417.041015625),
        (465.1421080732791, 3222.4052734375),
        (470.1669593722212, 8621.81640625),
        (475.23989190282134, 3369.073974609375),
```

(continues on next page)

(continued from previous page)

```
        (493.27465300375036, 2725.885986328125),
        (496.0077303201583, 8604.0830078125),
    ]
    print('{0:-^100}'.format('Library generation'))
    lib = pyqms.IsotopologueLibrary(
        molecules        = [ 'DDSPDLPK' ],
        charges          = [ 2 ],
        metabolic_labels = None,
        fixed_labels     = None,
        verbose          = True
    )
    print('{0:-^100}'.format('Library generation'))

    results = lib.match_all(
        mz_i_list = peak_list,
        file_name = 'BSA_test',
        spec_id   = 1165,
        spec_rt   = 29.10,
        results   = None
    )
    for key, i, entry in results.extract_results():
        p = pymzml.plot.Factory()
        label_mz_error = []
        label_i_error  = []
        measured_peaks = []
        matched_peaks  = []
        peak_info = ddict(list)
        # pprint.pprint(entry.peaks)
        for measured_mz, measured_intensity, relative_i, calculated_mz, calculated_
↪intensity in entry.peaks:
            if measured_mz is not None:
                measured_peaks.append(
                    (
                        measured_mz,
                        measured_intensity
                    )
                )
                matched_peaks.append(
                    (
                        calculated_mz,
                        calculated_intensity * entry.scaling_factor
                    )
                )
                mz_error = (measured_mz - calculated_mz) / ( measured_mz * 1e-6 )
                label_mz_error.append(
                    (
                        calculated_mz,
                        '{0:5.3f} ppm m/z error'.format(
                            mz_error
                        )
                    )
                )
                scaled_intensity = calculated_intensity * entry.scaling_factor
                rel_i_error  = abs(measured_intensity - scaled_intensity) / scaled_
↪intensity

                peak_info['measured peaks'].append(measured_mz)
```

(continues on next page)

(continued from previous page)

```
                peak_info['theoretical peaks'].append(calculated_mz)
                peak_info['relative intensity'].append(relative_i)
                peak_info['scaled matched peaks'].append( calculated_intensity *␣
↪entry.scaling_factor )
                peak_info['mz error'].append( mz_error)
                peak_info['i error'].append(  rel_i_error )


                if rel_i_error > 1:
                    rel_i_error = 1

                label_i_error.append(
                    (
                        calculated_mz,
                        '{0:5.3f} rel. intensity error'.format(
                            rel_i_error
                        )
                    )
                )



        mz_only  = [ n[0] for n in measured_peaks ]
        mz_range = [ min(mz_only)-1, max(mz_only)+1 ]
        peptide = results.lookup['formula to molecule'][key.formula][0]
        p.newPlot(
            header = 'Formula: {0}; Peptide: {1}; Charge: {2}\n Amount: {3:1.3f};␣
↪Score: {4:1.3f}'.format(
                key.formula,
                peptide,
                key.charge,
                entry.scaling_factor,
                entry.score
            ),
            mzRange = mz_range
        )
        p.add(
            measured_peaks,
            color = (0, 0, 0),
            style = 'sticks'
        )
        p.add(
            matched_peaks,
            color = (0, 200, 0),
            style = 'triangles'
        )
        p.add(
            label_mz_error,
            color = (255, 0, 0),
            style = 'label_x'
        )
        p.add(
            label_i_error,
            color = (255, 0, 0),
            style = 'label_x'
        )
```

(continues on next page)

(continued from previous page)

```
        plot_name = os.path.join(
            os.pardir,
            'data',
            'Score_visualization_Peptide_{1}_Charge_{2}.xhtml'.format(
                key.file_name,
                peptide,
                key.charge
            )
        )
        p.save(
            filename = plot_name,
            mzRange  = mz_range
        )
        print(
                'Plotted file {0}'.format(
                    plot_name
                )
            )
        # print(entry)
        print('Match info')
        for key, value_list in sorted(peak_info.items()):
            print(key)
            print('[{0}]'.format(','.join([str(n) for n in value_list])))
            print()
    return


if __name__ == '__main__':
    main()
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

C

D

E

H

I

M

N

P

R

S

U

W