pyOpenMS: A Python-based interface to the OpenMS mass-spectrometry algorithm library

Article *in* Proteomics · January 2014

DOI: 10.1002/pmic.201300246 · Source: PubMed

Hannes Luc Röst
Stanford University
24 PUBLICATIONS **1,395** CITATIONS
SEE PROFILE

Uwe Schmitt
ETH Zurich
18 PUBLICATIONS **208** CITATIONS
SEE PROFILE

Lars Malmström
ETH Zurich
86 PUBLICATIONS **2,308** CITATIONS
SEE PROFILE

TECHNICAL BRIEF

pyOpenMS: A Python-based interface to the OpenMS mass-spectrometry algorithm library

Hannes L. Röst[1,2]*, *Uwe Schmitt*[3]*, *Ruedi Aebersold*[1,4,5] *and Lars Malmström*[1]

[1] Department of Biology, Institute of Molecular Systems Biology, ETH Zurich, Zurich, Switzerland
[2] Ph.D. Program in Systems Biology, University of Zurich and ETH Zurich, Zurich, Switzerland
[3] Mineway GmbH, Saarbrücken, Germany
[4] Competence Center for Systems Physiology and Metabolic Diseases, Zurich, Switzerland
[5] Faculty of Science, University of Zurich, Zurich, Switzerland

pyOpenMS is an open-source, Python-based interface to the C++ OpenMS library, providing facile access to a feature-rich, open-source algorithm library for MS-based proteomics analysis. It contains Python bindings that allow raw access to the data structures and algorithms implemented in OpenMS, specifically those for file access (mzXML, mzML, TraML, mzIdentML among others), basic signal processing (smoothing, filtering, de-isotoping, and peak-picking) and complex data analysis (including label-free, SILAC, iTRAQ, and SWATH analysis tools). pyOpenMS thus allows fast prototyping and efficient workflow development in a fully interactive manner (using the interactive Python interpreter) and is also ideally suited for researchers not proficient in C++. In addition, our code to wrap a complex C++ library is completely open-source, allowing other projects to create similar bindings with ease. The pyOpenMS framework is freely available at https://pypi.python.org/pypi/pyopenms while the autowrap tool to create Cython code automatically is available at https://pypi.python.org/pypi/autowrap (both released under the 3-clause BSD licence).

Received: June 24, 2013
Revised: September 11, 2013
Accepted: October 11, 2013

Keywords:
Bioinformatics / Data mining / Mass spectra / Proteome maps / Python / Software

Additional supporting information may be found in the online version of this article at the publisher's web-site

Computational data analysis in the field of high-throughput, LC-MS/MS based proteomics can be multifacetted and diverse and in many cases must be tailored to a specific set of samples or experimental condition. This is due to the availability of a wide range of options at each step of a proteomics analysis: Whole proteomes can be measured directly, fractionated using different techniques or specific sub proteomes may be selectively enriched (using e.g. affinity-purification, cell surface capture, phosphopeptide enrichment, and others). For quantification, different isotopic labeling methods are available (e.g. ICAT, SILAC, iTRAQ, TMT (where TMT is tandem mass tags)) or a label-free strategy can be chosen. At the data acquisition step, different types of instruments either support data-dependent acquisition or data-independent acquisition (DIA) while others support targeted data acquisition, for example, SRM. Finally, the overall analysis strategy will depend (in addition to the options chosen in upstream data acquisition steps) also on the choice between different data processing options including database search engines, spectral library searching, or a targeted analysis (for SRM or DIA) [1, 2]. Frequently, therefore, an optimal data analysis strategy has to be developed or adapted for a specific study or dataset.

OpenMS is an open-source, C++ based software library that accommodates the need for flexibility in the analysis of

Correspondence: Dr. Lars Malmström, Department of Biology, Institute of Molecular Systems Biology, ETH Zurich, CH-8093 Zurich, Switzerland
E-mail: lars@imsb.biol.ethz.ch
Fax: +41-44-633-10 51

Abbreviation: DIA, data-independent acquisition

*These authors contributed equally to this work.

proteomics data by providing over 100 executable tools that perform different steps in the computational data analysis workflow, supporting nearly all file formats and database search engines commonly used in MS-based proteomics [3–5]. It has proven effective in analyzing datasets generated from isotopically labeled as well as label-free samples [6, 7]. Recently, support for targeted proteomics data such as SRM and DIA (SWATH-MS) has been added (H. L. Röst et al., paper accepted in *Nature Biotechnology* 2013). While being a very useful tool for data analysis, the development of novel algorithms, or the combination of existing algorithms into complex data analysis pipelines requires extensive knowledge of C++ programming and the OpenMS software development process.

To make flexible generation of data analysis pipelines as well as custom algorithm development accessible to a broader community of proteomics researchers, we have developed pyOpenMS, a Python-based wrapper to access the OpenMS library. Taking advantage of the OpenMS model—which compiles the computing-intensive algorithms into a shared library—we wrapped the exposed application programming interface of the library using Cython and the novel autowrap tool developed for this project, providing full access to OpenMS objects and functions from Python. Python is a mature scripting language with high acceptance in the biology community, already supporting many tasks from biological sequence handling with Biopython [8] to structural modeling and visualization with PyMOL [9] and PyRosetta [10], to numerical computation and advanced plotting with NumPy/SciPy [11], rpy2 [12], and matplotlib [13]. The tools described in this paper thus combine the power of OpenMS with Python, a mature, easy-to-learn, cross-platform scripting language that is especially suitable for beginners.

We used Cython (C-Extension for Python) and a newly developed software called autowrap to wrap C++ classes and functions and make them available from within Python. Wrapping a class starts by creating a `.pxd` file containing only the class and function declarations of the code to be wrapped. Autowrap then generates a corresponding `.pyx` file, handling the memory management using boost shared pointers, type-checking of Python-input and conversion of more complex Standard Template Library data structures (as well as OpenMS-internal data structures like StringList or DataValue) to and from Python automatically. This ensures consistent code quality and error handling while allowing for very fast and easy wrapping of new classes; the autowrap tool is available as a standalone software at the Python Package Index PyPI under the 3-clause BSD licence (https://pypi.python.org/pypi/autowrap). In the next step, Cython parses the generated `.pyx` file as well as the associated `.pxd` function declaration files (specifying which functions will be exposed) and generates a `.cpp` file that is then compiled into a Python-module (see Fig. 1 for our workflow). In this manner, we have wrapped over 4100 C++ method calls in Python. The whole process is tightly integrated with the OpenMS build process and is currently executed nightly on the newest SVN checkout and a battery of over 200 tests are automatically executed to ensure constant compatibility with the newest C++ source code.

The pyOpenMS Python bindings provide a rich set of features that include:

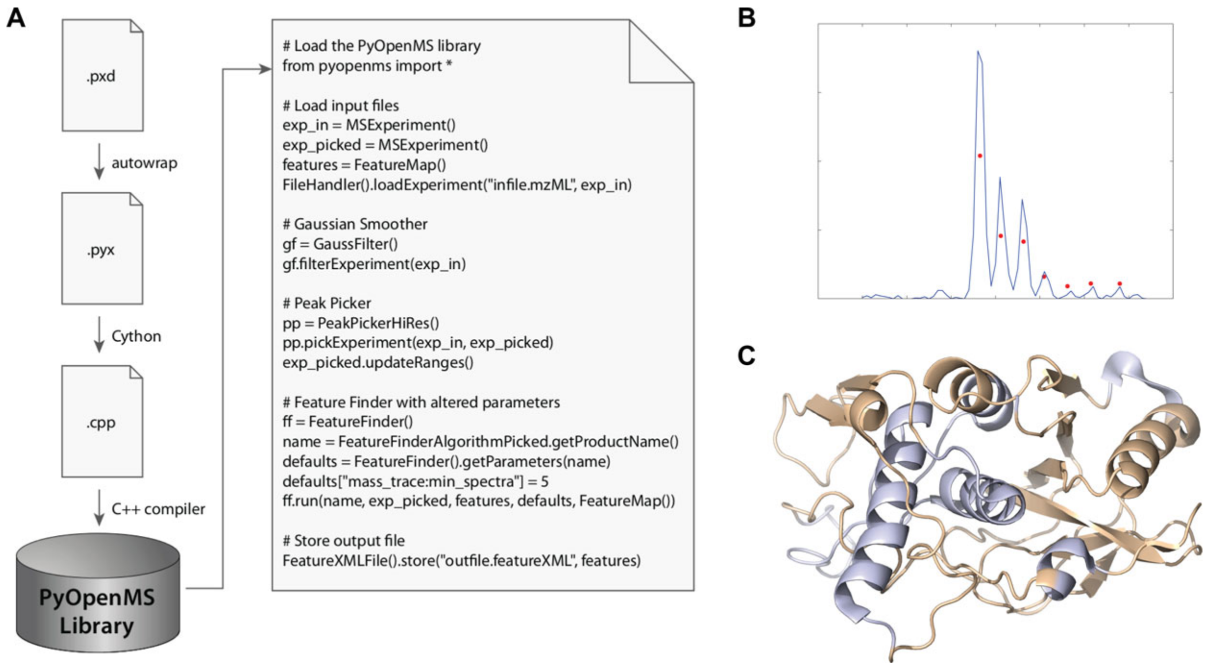
File handling: pyOpenMS provides fully standard-compliant readers and writers of the file formats developed by the proteomics standards initiative [14], including mzML, TraML, mzIdentML as well as the upcoming mzQuantML standard [15–17]. The underlying raw data are conveniently provided in NumPy arrays for fast data handling and processing with tools outside pyOpenMS, for example, allowing plotting with matplotlib or data processing using a numeric library like `scipy.signal`. In total, pyOpenMS supports over 30 different file formats including PepXML, ProtXML, trafoXML, IdXML, featureXML, consensusXML, mzXML, and many more.

Basic functionality and signal processing: Most basic signal processing algorithms implemented in OpenMS have been wrapped in pyOpenMS, including smoothing, baseline filtering, and peak-picking algorithms. Furthermore, functions that perform common mass spectrometric tasks such as TOF calibration, de-isotoping and chromatogram extraction, and a set of spectral filters are also available.

Complex analysis tools: Most interestingly, pyOpenMS can handle the complex analysis tools provided in OpenMS and exposes their corresponding application programming interfaces to Python—which are in most cases very simple, requiring only input and output data objects as well as a parameter handling object (see Fig. 1 for an example of such a workflow). We have thus wrapped the function calls performed by the OpenMS SILACAnalyzer, OpenSwathAnalyzer, iTRAQAnalyzer, FeatureFinderCentroided, and FeatureFinderSuperHirn (for 2D feature detection in LC-MS/MS maps) as well as several other complex tools.

To illustrate the power of pyOpenMS, we have created several demonstration applications that show how pyOpenMS can be integrated within the Python programming environment to rapidly produce high-quality results (all Python code is provided in the Supporting Information). In Fig. 1 we show three such applications: (a) a simple workflow that performs data smoothing and de-isotoped feature quantification using pyOpenMS, (b) the demonstration of a novel peak-picking implementation using Fast Fourier Transform in SciPy [11], and (c) a three dimensional visualization of a protein structure overlaid with the peptides identified in a LC-MS/MS experiment (in gold) using pyMOL [9].

We next applied pyOpenMS to a real biological problem, investigating the issue of correct phosphosite assignment in a mass spectrometric dataset. In phospho-proteomics it is often possible to accurately identify the peptide sequence and the number of phosphorylations, however, reporting the localization of the phosphorylated residue is considered nontrivial (however, this is often crucial knowledge since the biological function may differ depending on the phosphorylated



```
# Load the PyOpenMS library
from pyopenms import *

# Load input files
exp_in = MSExperiment()
exp_picked = MSExperiment()
features = FeatureMap()
FileHandler().loadExperiment("infile.mzML", exp_in)

# Gaussian Smoother
gf = GaussFilter()
gf.filterExperiment(exp_in)

# Peak Picker
pp = PeakPickerHiRes()
pp.pickExperiment(exp_in, exp_picked)
exp_picked.updateRanges()

# Feature Finder with altered parameters
ff = FeatureFinder()
name = FeatureFinderAlgorithmPicked.getProductName()
defaults = FeatureFinder().getParameters(name)
defaults["mass_trace:min_spectra"] = 5
ff.run(name, exp_picked, features, defaults, FeatureMap())

# Store output file
FeatureXMLFile().store("outfile.featureXML", features)
```

Figure 1. The pyOpenMS workflow and several complex sample analyses achievable using pyOpenMS. (A) The workflow we used to create the Python bindings is depicted on the left side. The only manual step is to write the `.pxd` function declaration files, which are then automatically wrapped using autowrap, then converted to C++ using Cython and finally compiled into a Python extension module. On the right side is a sample Python script that imports the extension module on the top and executes a simple workflow to load data from a proteomics experiment, then processes the data including smoothing and peak-picking and finally extracts de-isotoped features using the `FeatureFinder` tool. (B) To illustrate the power of prototyping algorithms in Python, we have written a small ($<$75 lines, see Supporting Information) Python script that implements an Fast Fourier Transform based lowpass filter using SciPy for peak-picking and visualizes the results using `matplotlib`. The raw sample data (taken from the OpenMS testdata set) is shown in blue and the picked peaks in red. (C) As another example of how pyOpenMS can be integrated with other powerful Python packages, we used PyMol to display the sequence coverage of the pdb structure 4D8B in a recent proteomics experiment performed in our lab: golden parts of the structure were covered by identified peptides (blue parts were not covered). See Supporting Information for our script.

residue). To evaluate different phosphorylation-localization algorithms, we used pyOpenMS to read a PepXML file and the corresponding raw spectra, then passed the spectrum with the matched identification to a scoring function (see Table 1 for the Python code fragment and the Supporting Information for the actual implementation comparing AScore [18] to a simplistic, spectrum-binning based approach implemented here as proof-of-concept). These applications show the ease with which novel ideas can be implemented within the pyOpenMS framework and how existing data structures can be read, combined, matched, and processed with minimal coding effort. pyOpenMS allows for rapid exploration of different analysis strategies with very little overhead—while at the same time the shared underlying data structures make

Table 1. **Phosphopeptide site assignment.** The following code fragment illustrates how pyOpenMS can be used to iterate through a set of peptide identifications and corresponding spectra to compute a phosphosite assignment probability for each identification (such as described in AScore). See Supporting Information for the actual implementation.

```
import pyopenms
[...]
pyopenms.PepXMLFile().load(pepxml_file, protein_ids, peptide_ids)
mass_tol = 0.5
for peptide_id in peptide_ids:
peptide_hit = peptide_id.getHits()[0] # retrieve best hit
spectrum = retrieveSpectrumForIdentification(peptide_id)
nr_sites = peptide_hit.getSequence().toString().count("Phospho")
ascore_hit = pyopenms.AScore().compute(peptide_hit, spectrum, mass_tol, nr_sites)
print ascore_hit.getMetaValue("AScore_1"), ascore_hit.getSequence().toString()
```

implementation of the finished algorithm in C++ straightforward.

In conclusion, pyOpenMS is a versatile Python-based implementation of the OpenMS functionality, allowing even novice users to create, adapt, and manage relatively complex workflows in Python with ease while expert programmers can benefit from the fast prototyping offered by the Python language. Thus, the user has the opportunity to write prototype code or whole analysis workflows coupled with a statistical analysis in the same script, while having direct access to the high-performance algorithms in the OpenMS C++ library. In addition to a mere wrapping of C++ function calls, we have adapted the Python objects to the "look and feel" of Python, providing iterators and direct attribute access for core classes—making it even easier to use the interface. This allows for new applications and an algorithmic as well as workflow development process that was previously closed to nonexperts in C++ programming. Finally, providing access to an well-established algorithmic library for proteomics data analysis such as OpenMS aligns with recently described efforts to establish Python in the proteomics community (e.g. pymzML and Pyteomics, which provide specific functionality for certain proteomics data analysis [19, 20]). With this, we hope that future projects are able to build upon these foundations and benefit from the large amount of already implemented functionality. The complete Python bindings as well as all source code, sample tools, and workflows are open source and accessible through the OpenMS SVN repository.

In addition to the Python-bindings for OpenMS described here, we also provide the open-source autowrap tool that allows fast and easy wrapping of C++ code for Python, thus facilitating similar projects in the future.

The authors would like to thank the developers of the OpenMS project for their support during this project and all their work and effort to integrate pyOpenMS into the nightly build system of OpenMS; specifically we would like to thank George Rosenberger, Stephan Aiche, and Oliver Kohlbacher.

Funding: This project was support by ETH Zurich, Department of Biology, within the frame of an IT-strategy initiative. H.R. was funded by ETH (ETH-30 11-2).

The authors have declared no conflict of interest.

References

[1] Aebersold, R., Mann, M., Mass spectrometry-based proteomics. *Nature* 2003, *422*, 198–207.

[2] Domon, B., Aebersold, R., Options and considerations when selecting a quantitative proteomics strategy. *Nature Biotechnology* 2010, *28*, 710–721.

[3] Sturm, M., Bertsch, A., Gröpl, C., Hildebrandt, A. et al., OpenMS – an open-source software framework for mass spectrometry. *BMC Bioinformatics* 2008, *9*, 163.

[4] Kohlbacher, O., Reinert, K., Gröpl, C., Lange, E. et al., TOPP – the OpenMS proteomics pipeline. *Bioinformatics* 2007, *23*, e191–e197.

[5] Bertsch, A., Gröpl, C., Reinert, K., Kohlbacher, O., OpenMS and TOPP: open source software for LC-MS data analysis. *Methods Mol Biol* 2011, *696*, 353–367.

[6] Weisser, H., Nahnsen, S., Grossmann, J., Nilse, L. et al., An Automated Pipeline for High-Throughput Label-Free Quantitative Proteomics. *J. Proteome Res.* 2013, *12*, 1628–1644.

[7] Nilse, L., Sturm, M., Trudgian, D., Salek, M. et al., SILACAnalyzer – A Tool for Differential Quantitation of Stable Isotope Derived Data. in: Masulli, F., Peterson, L., Tagliaferri, R. (eds.), *Computational Intelligence Methods for Bioinformatics and Biostatistics,* vol. 6160 *of Lecture Notes in Computer Science*, Chapter 4, Springer, Berlin Heidelberg, 2010, pp. 45–55.

[8] Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A. et al., Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* 2009, *25*, 1422–1423.

[9] Delano, W. L., The PyMOL Molecular Graphics System version 1.4.1-3 Schrödinger, LLC, 2002. Available at: http://www.pymol.org

[10] Chaudhury, S., Lyskov, S., Gray, J. J., PyRosetta: a script-based interface for implementing molecular modeling algorithms using Rosetta. *Bioinformatics* 2010, *26*, 689–691.

[11] Jones, E., Oliphant, T., Peterson, P. et al., SciPy: Open source scientific tools for Python, 2001.

[12] rpy2 Development Team 2004, http://rpy.sourceforge.net/rpy2.html.

[13] Hunter, J. D., Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng.* 2007, *9*, 90–95.

[14] Orchard, S., Hermjakob, H., Apweiler, R., The proteomics standards initiative. *Proteomics* 2003, *3*, 1374–1376.

[15] Martens, L., Chambers, M., Sturm, M., Kessner, D. et al., mzML – a community standard for mass spectrometry data. *Mol. Cell. Proteomics* 2011, *10*, R110.000133.

[16] Deutsch, E. W., Chambers, M., Neumann, S., Levander, F. et al., TraML – A Standard Format for Exchange of Selected Reaction Monitoring Transition Lists. *Mol. Cell. Proteomics* 2012, *11*, R111.015040.

[17] Jones, A. R., Eisenacher, M., Mayer, G., Kohlbacher, O. et al., The mzIdentML Data Standard for Mass Spectrometry-Based Proteomics Results. *Mol. Cell. Proteomics* 2012, *11*, M111.014381.

[18] Beausoleil, S. A., Villén, J., Gerber, S. A., Rush, J., Gygi, S. P., A probability-based approach for high-throughput protein phosphorylation analysis and site localization. *Nat. Biotechnol.* 2006, *24*, 1285–1292.

[19] Goloborodko, A., Levitsky, L., Ivanov, M., Gorshkov, M., Pyteomics – a Python Framework for Exploratory Data Analysis and Rapid Software Prototyping in Proteomics. *J. Am. Soc. Mass Spectrom.* 2013, *24*, 301–304.

[20] Bald, T., Barth, J., Niehues, A., Specht, M. et al., pymzML – Python module for high-throughput bioinformatics on mass spectrometry data. *Bioinformatics* 2012, *28*, 1052–1053.