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, CH-8093 Zurich, Switzerland*
[2]*Ph.D. Program in Systems Biology, University of Zurich and ETH Zurich, CH-8057 Zurich, Switzerland*
[3]*mineway GmbH, 66121 Saarbrücken, Germany*
[4]*Competence Center for Systems Physiology and Metabolic Diseases, CH-8093 Zurich, Switzerland*
[5]*Faculty of Science, University of Zurich, CH-8057 Zurich, Switzerland*

(Dated: October 19, 2015)

Abstract

pyOpenMS is an open-source, Python-based implementation of the C++ OpenMS library, providing facile access to a feature-rich, open-source algorithm library for mass-spectrometry 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 available under the 3-clause BSD licence).

Computational data analysis in the field of high-throughput LC-MS/MS based proteomics can be very 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) or a label-free strategy can be chosen. At the data acquisition step, different types of instruments either support data-dependent acquisition (DDA) or data-independent acquisition (DIA) while others support targeted data acquisition by selected reaction monitoring (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].

OpenMS is an open-source, C++ based software library that accommodates the need for flexibility in the analysis of 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 mass spectrometry-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 has been added [8]. 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 custom algorithm development and the flexible generation of data analysis pipelines 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 API 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 [9] to structural modeling and visualization with PyMOL [10] and PyRosetta [11], to numerical computation and advanced plotting with numpy/scipy [12] and matplotlib [13]. We were thus able to 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 automatically 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 automatically generates a corresponding .pyx file, handling the memory management using boost shared

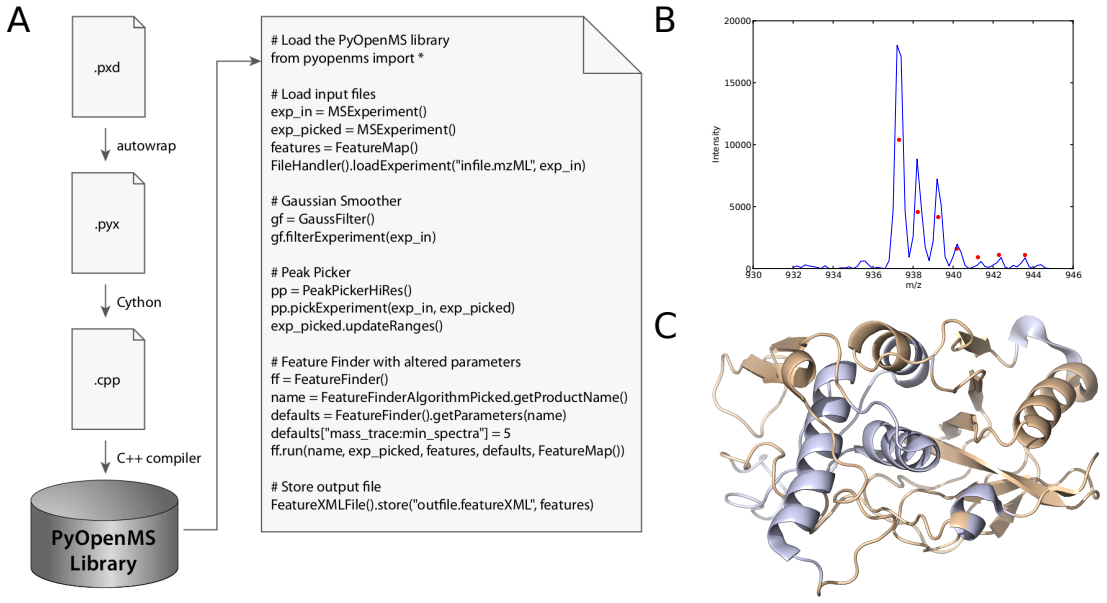

FIG. 1: **The pyOpenMS workflow and several complex examples achievable using pyOpenMS A)** The workflow we used to create the Python bindings is depicted on the left side. The only manual step is to write the `.pxd` function declaration files, which are then automatically wrapped using autowrap, then 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 supplemental material) Python script that implements an Fast Fourier Transform (FFT)-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 supplemental material for our script.

pointers, type-checking of Python-input and conversion of more complex STL datastructures (as well as OpenMS-internal datastructures 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 (basically specifying which functions should be wrapped) and generates a .cpp file that is then compiled into a Python-module (see Figure 1 for our workflow). In this manner, we have wrapped over 4000 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 one two hundred tests are automatically executed to ensure constant compatibility with the newest C++ source code.

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

File handling: pyOpenMS provides fully standard-compliant readers and writers of the file formats developed by the Proteomics Standards Initiative (PSI) [14], including mzML, TraML, mzIdentML as well as the upcoming mzQuantML standard [15–17]. The underlying raw data is 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 APIs to Python - which are in most cases very simple, requiring only input and output data objects as well as a parameter

handling object (see Figure 1 for an example workflow). We have thus wrapped the function calls performed by the OpenMS SILACAnalyzer, OpenSwathAnalyzer, iTRAQAnalyzer, FeatureFinderCentroided and FeatureFinderSuperHirn (for 2-dimensional 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 which show how pyOpenMS can be integrated within the Python programming environment to rapidly produce high-quality results (all Python code is provided in the supplemental material). In Figure 1 we show three such applications: i) starting with a simple workflow that performs data smoothing and de-isotoped feature quantification using pyOpenMS ii) to the demonstration of a novel peak-picking implementation using Fast Fourier Transform (FFT) in `scipy` [12], to a three dimensional visualization of a protein structure overlaid with the peptides identified in a LC-MS/MS experiment in gold using pyMOL [10].

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 that was previously closed to non-experts in C++ programming. Finally, providing access to a mature algorithmic library for proteomics data analysis aligns with recently described efforts, such as pymzML and Pyteomics, to use Python for proteomics data analysis [18, 19]. 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.

Contributions

H.R. & U.S. designed, implemented and executed the C++ code and the analysis workflow. H.R. & U.S. & L.M. & R.A. wrote the manuscript. L.M. designed and supervised the study.

Acknowledgements

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 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.

[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*.

[4] Kohlbacher, O., Reinert, K., Gröpl, C., Lange, E. et al., TOPP – the OpenMS proteomics pipeline. *Bioinformatics* 2007, *23*, e191–197.

[5] Bertsch, A., Gröpl, C., Reinert, K., Kohlbacher, O., OpenMS and TOPP: open source software for LC-MS data analysis. *Methods in molecular biology (Clifton, N.J.)* 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. *Journal of Proteome Research* 2013, *12*.

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

[8] Röst, H., Rosenberger, G., Navarro, P., Gillet, L. et al., OpenSWATH: Automated, targeted analysis of data-independent acquisition (DIA) MS-data. *under revision* 2013.

[9] 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.

[10] Delano, W. L. 2002, The PyMOL Molecular Graphics System.

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

[12] Jones, E., Oliphant, T., Peterson, P., Others 2001-, SciPy: Open source scientific tools for Python.

[13] Hunter, J. D., Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 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. *Molecular & cellular proteomics : MCP* 2011, *10*.

[16] Deutsch, E. W., Chambers, M., Neumann, S., Levander, F. et al., TraML—A Standard Format for Exchange of Selected Reaction Monitoring Transition Lists. *Molecular & Cellular Proteomics* 2012, *11*.

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

[18] Goloborodko, A., Levitsky, L., Ivanov, M., Gorshkov, M., Pyteomics—a Python Framework for Exploratory Data Analysis and Rapid Software Prototyping in Proteomics. *Journal of the American Society for Mass Spectrometry* 2013, *24*, 301–304.

[19] 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.