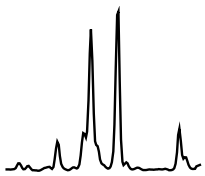

# PyMS User Guide



PyMS version 1.0

A Python toolkit for processing of chromatography–mass spectrometry data

# Contents

# Introduction

## 1.1 About PyMS

PyMS is a Python toolkit for processing of chromatography–mass spectrometry data. The main idea behind PyMS is to provde a framework and a set of components for rapid development and testing of methods for processing of chromatography–mass spectrometry data. An important objective of PyMS is to decouple processing methods form visualization and the concept of interactive processing. This is useful for high-throughput processing tasks and when there is a need to run calculations in the batch mode.

PyMS is highly modular and consists of several sub-packages written in Python programming language [1]. PyMS is released as open source, under the GNU Public License version 2.

## 1.2 PyMS installation

There are many ways to install PyMS depending your computer configuration and preferences. The recommended way install PyMS is to compile Python from sources and install PyMS within the local Python installation. This procedure is described in detail below.

PyMS has been developed on Linux, and a detailed installation instructions for Linux are given below. Installation on any Unix-like system should be similar. We have not tested PyMS under Microsoft Windows. If you are using/testing PyMS under Windows please let us know.

### 1.2.1 Downloading PyMS source code

PyMS source code resides on Google Code servers, and can be accessed from the following URL: http://code.google.com/p/pyms/. Under the section "Source" one can find the instructions for downloading the source code. The same page provides the link under "This project's Subversion repository can be viewed in your web browser" which allows one to browse the source code on the server without actually downloading it.

Google Code maintains the source code by the program called 'subversion' (an open-source version con-

trol system). To download the source code one needs to use the subversion client program called 'svn'. The 'svn' client exists for all mainstream operating systems[1], for more information see http://subversion.tigris.org/. The book about subversion is freely available on-line at http://svnbook.red-bean.com/. Subversion has extensive functionality. However only the very basic functionality is needed to download PyMS source code.

If the computer is connected to the internet and the subversion client is installed, the following command will download the latest PyMS source code:

```
$ svn checkout http://pyms.googlecode.com/svn/trunk/ pyms
A    pyms/Peak
A    pyms/Peak/__init__.py
A    pyms/Peak/List
A    pyms/Peak/List/__init__.py
.....
Checked out revision 71.
$ ls -CF
pyms/
$
```

### 1.2.2 Installing Python locally

Download the latest Python from `www.python.org`. At the time of writing this was `Python-2.5.2.tgz`. Place the Python tarball in some temporary directory, and unpack it:

```
$ tar xvfz Python-2.5.2.tgz
Python-2.5.2/
Python-2.5.2/Include/
Python-2.5.2/Include/pymem.h
Python-2.5.2/Include/compile.h
Python-2.5.2/Include/pyexpat.h
[..output deleted..]
```

Change to `Python-2.5.2/`, configure and compile Python:

```
$ cd Python-2.5.2
$ ./configure --prefix=/home/likic/programs/python2.5.2
[..output deleted..]
$ mkdir -p /home/likic/programs/python2.5.2
```

[1] For example, on Linux CentOS 4 we have installed the RPM package 'subversion-1.3.2-1.rhel4.i386.rpm' to provide us with the subversion client 'svn'.

The above command specifies that Python will be installed in the directory `/home/likic/programs/python`. The last command creates this directory in preparation for installation. Python is compiled and installation as follows:

```
$ make
[..output deleted..]
$ make install
[..output deleted..]
```

To check the installation directory:

```
$ ls -CF  /home/likic/programs/python2.5.2
bin/  include/  lib/  share/
```

The compiled Python executable is `/home/likic/programs/python2.5.2/bin/python`:

```
$ /home/likic/programs/python2.5.2/bin/python
Python 2.5.2 (r252:60911, Aug  1 2008, 07:57:44)
[GCC 4.1.1 20070105 (Red Hat 4.1.1-52)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

This completes Python installation.

## 1.3 Minmax peak detector

### 1.3.1 Introduction

Minmax peak detector is the simplest kind of a peak finding algoritam for TIC. It operates by finding peak maxima, and then attempting to determine peak boundaries. Cursory evidence suggests that gives results similar to the ChemStation peak detection, but this was not examined rigorously. The purpose of this algorithm is to provide an example of how 1D peak pickin can be implemented in PyMS. *At present the Minmax algorithm was not properly tested, do not use it for critical publication quality results.*

### 1.3.2 A brief description of the algorithm

Many peak detection algorithms are used in practice to process GC/LC-MS data, but only a few are fully documented, most notable those of open source projects MZmine [2] and XCMS [3]. MZmine detects peaks by finding local maxima of a certain width [2]. In XCMS peaks are detected by using an empirical signal-to-noise cutoff after matched filtration with a second-derivative Gaussian [3]. PyMS peak detection

procedure was developed in-house, and relies on finding local maxima and local minima in the signal, followed by a subsequent refinement of peak left and right boundaries. Peak detection depends on two input parameters: window width over which a peak is expected to be a global maximum, and the scaling factor $S$ used to calculate the intensity threshold $S\sigma$ which must be exceeded at the peak apex. The noise level $\sigma$ is estimated prior to peak detection by repeatedly calculating median absolute deviation (MAD – a robust estimate of the average deviation) over randomly placed windows and taking the minimum. A detailed description of procedures for peak detection follows.

1. **Extracting local maxima**. Initially, an ordered list of local maxima in the signal with an intensity larger than a threshold is compiled. Two input parameters are specified by the user: the width of a window over which the peak is required to be a global maximum ($W$); and (2) the scaling factor $S$ used to calculate intensity threshold $S\sigma$, where $S$ is the noise level estimated previously (defaults: $W = 2$ data points, $S = 10$). User specified window is centered on each point of the signal, and the point is deemed to be a local maximum if the following is satisfied:

   (a) It is equal or greater than all of the points within the window W.

   (b) It is greater than at least one point in the half-window interval to the left, and at least one point in the half-window interval to the right.[2]

   (c) Any point closer to the edge of the signal than half-window is rejected.

   Intensity at each local maxima is tested, and those that have the intensity below the threshold N*S are rejected. Accepted local maxima are compiled into a list.

2. **Determination of peak left/right boundaries**. For each local maxima (base maximum) the stretch of the signal between itself and the next local maximum on either side is extracted. These two signal slices are searched for the first local minimum in the direction away from the base maximum point itself. The local maxima are defined in a very similar manner as the local maxima in the previous step. A point is deemed to be a local maximum if:

   (a) It is equal or smaller than all the points within the window W.

   (b) It is smaller than at least one point in the half-window interval to the left, and at least one point in the half-window interval on the right.

   (c) Any point closer to the edge of the slice than half-window is rejected. This has the effect that the boundary point cannot approach next peak's apex closer than half-window.

   (d) If no minimum point is found, set the boundary point to the point furtherest away from the base maximum, but outside to the half-window range of the adjacent peak.

3. **Elimination of peak overlaps**. In spectra dense with peaks peak boundaries as found in the step (2) may overlap due to the effect of user supplied window. The list of pre-peaks is searched for overlapping peaks. In overlapping peaks the right boundary of the lower retention time peak overlaps with the left boundary of the higher retention time peak. The overlapping boundaries are resolved by finding the point of minimum intensity between the two peaks (the split point). The peak boundaries are set to one point to the left from the split point for the right boundary of the

[2] This is to reject points within intervals of uniform intensity

lower retention time peak, and to one point to the right from the split point for the left boundary of the higher retention time peak.

4. **Correction for long tails**. In this step peak boundaries are adjusted to remove stretches of near-uniform intensities (i.e. long tails). Each peak is divided at the apex into two halves, and each half is processed individually in the boundary-to-apex direction. A line is fitted through $M$ points from the boundary in the least-squares sense. Prior to calculating the angle between the line and the retention time axis, the rise in intensity is normalized with the intensity at the peak apex. If this angle is below the user specified cutoff ($Q$) the boundary point is dropped, and the process is repeated. This adjustment is repeated until the best fit through $M$ points from the boundary gives an angle greater than the cutoff. The parameters $M$ and $Q$ are user specified (defaults: $M = 3$, $Q = 1.0°$).

# Using PyMS

## 2.4 Reading GC-MS data

### 2.4.1 ANDI-MS data format

The field of chromatography–mass spectrometry is ruled by proprietary, closed data formats. Anything that comes even closse to an open standard is the incomplete and now obsolete ANDI-MS data format.

ANDI-MS data format stands for Analytical Data Interchange for Mass Spectrometry, and was developed for the description of mass spectrometric data developed in 1994 by Analytical Instrument Association. ANDI-MS is essentially a recommendation, and it is up to individual vendors of mass spectrometry processing software to implement "export to ANDI-MS" feature in their software. Furthermore, it is vendor's good will to implement ANDI-MS specifications properly. Because of these limitations it is difficult to be certain that one can properly read ANDI-MS files from a particular vendor without testing this first.

### 2.4.2 Reading the data in PyMS

The PyMS package pyms.IO provides capabilities to read the raw GC-MS data stored in the ANDI-MS format. The function IO.ANDI.ChemStation() provides the interface to ANDI-MS data files saved from Agilent ChemStation software. The name is a reminder that this function has been reasonably tested only on the data exported from Agilent ChemStation.

In an interactive session from Python, the ANDI-MS file can be loaded in the memory as follows:

```
>>> from pyms import IO
>>> data = IO.ANDI.Class.ChemStation('0510_217.CDF')
 -> Processing netCDF file '0510_217.CDF'
    [ 2784 scans, masses from 50 to 550 ]
>>>
```

Where '0510_217.CDF' is the name of the GC-MS data saved in the ANDI-MS format from the Agilent

ChemStation software. The above command creates the object 'data' which is an *instance* of the class IO.ANDI.ChemStation. The instance 'data' has several attributes and methods associated with it:

- `get_filename()` – Returns the name of the file from which the data was loaded. Usage example:

  ```
  >>> data.get_filename()
  '0510_217.CDF'
  ```

- `get_ic_at_index(i)` – Returns an IonChromatogram object at index i. For example, to get the first ion chromatogram from the data matrix:

  ```
  >>> ic = data.get_ic_at_index(1)
  ```

  An IonChromatogram object is the one dimensional time vector containing mass intensities. One often deals with two types of IonChromatogram objects: ion chromatograms at particular m/z value (for example, ion chromatograms at m/z = 65), or total ion chromatograms (TICs), which contain the sum of intensities for all masses at any given time point. The nature of an IonChromatogram object can be revealed by the content of the attribute '_mass', which is set to None if the ion chromatogram is TIC; otherwise it contains the m/z value of the ion chromatogram. Continuing the previous example:

  ```
  >>> ic._mass
  51
  ```

  This shows that the first ion chromatogram in the data file is for m/z = 51.

- `get_ic_at_mass(mz)` – Returns an IonChromatogram object corresponding to given m/z. For example, to get the ion chromatogram that corresponds to m/z = 73:

  ```
  >>> ic = data.get_ic_at_mass(73)
  >>> ic._mass
  73
  ```

- `get_intensity_matrix()` – Returns the entire data matrix, i.e. time vs m/z as numarray object. Usage example:

  ```
  >>> im = data.get_intensity_matrix()
  >>> len(im)
  2784
  >>> len(im[0])
  501
  ```

  This data matrix contains 2784 time points (MS scans) and each time point corresponds to a mass spectrum of 501 m/z points.

# Bibliography

[1] Python. http://www.python.org.

[2] Katajamaa M, Miettinen J, and Oresic M. MZmine: toolbox for processing and visualization of mass spectrometry based molecular profile data. *Bioinformatics*, 22(5):634–636, 2006.

[3] Smith CA, Want EJ, O'Maille G, Abagyan R, and Siuzdak G. XCMS: processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching, and identification. *Anal Chem*, 78(3):779–87, 2006.