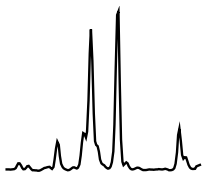

PyMS User Guide

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 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.

There are four parts of the pyms project:

- pyms – The PyMS code
- pyms-docs – The PyMS documentation
- pyms-test – Examples of PyMS use
- pyms-data – Data used in pyms-test

Each part is a separate project on Google Code that can be downloaded separately.

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.

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 control 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 Minimal PyMS installation

The minimal PyMS installation involves dowloading the PyMS code and installing the pycdf package. Download the pycdf package from `http://pysclint.sourceforge.net/pycdf/`, and follow the installation instructions that come with it. To test your pycdf installation:

```
$ python
Python 2.5.1 (r251:54863, Jul 10 2007, 08:23:47)
[GCC 4.1.1 20070105 (Red Hat 4.1.1-52)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pycdf
>>>
```

[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'.

If the pycdf is installed properly the above import statement will produce no error messages.

The PyMS code (ie. directory pyms/) can be placed anywhere as long as the Python interpreter is made aware where it is (see Python documentation for details on how to make packages visible to Python interpreter). In example, we keep the pyms code in `/home/current/proj/PyMS/` (ie. the PyMS code is `/home/current/proj/PyMS/pyms/`). Since this is a non-standard location for Python packages, before running PyMS we set the following in Python:

```
import sys
sys.path.append("/home/current/proj/PyMS/")
```

This makes Python aware that some packages may be in `/home/current/proj/PyMS/`. Therefore to check the PyMS installation one might try the following:

```
$ python
Python 2.5.1 (r251:54863, Jul 10 2007, 08:23:47)
[GCC 4.1.1 20070105 (Red Hat 4.1.1-52)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path.append("/home/current/proj/PyMS/")
>>> from pyms.IO.ANDI.Class import ChemStation
>>>
```

The last command has loaded the class 'ChemStation' from PyMS. Since this did not produce any error messages the PyMS was accessible (and moreover, that pycdf is properly installed, since the class 'ChemStation' uses pycdf).

For examples on PyMS use see Chapter 2.

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 σ 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.

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

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 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

2.1 Introduction

This chapter demonstrates main functions of PyMS in a tutorial like manner. The data files used in the examples are provided in the project 'pyms-data'. The commands executed interactively are grouped together by example, and provided as Python scripts in the project 'pyms-test'.

The setup used in the examples below is as follows. The projects 'pyms', 'pyms-test', 'pyms-docs', and 'pyms-data' were downloaded in the directory `/home/current/proj/PyMS`. In the project 'pyms-test' there is a directory corresponding to each example coded with the example number (ie. `pyms-test/01/` corresponds to Example 1). In each example directory there is a script named 'proc.py' which contains the commands given in the example. Provided that the paths to 'pyms' and 'pyms-data' are set properly, these scripts could be run by simply:

$ python proc.py

Before running each example the Python interpreter was made aware of the PyMS location with the following commands:

```
import sys
sys.path.append("/home/current/proj/PyMS/")
```

For brevity these commands will not be shown in the examples below, but they are included in 'pyms-test' example scripts. The above path need to be adjusted to match your own location of pyms.

All data files (raw data files, peak lists etc) used in the example below can be found in 'pyms-data'.

2.2 Example 1: Reading of GC-MS data and basic manipulations with data

2.2.1 Reading ChemStation GC-MS data into PyMS

This example is in pyms-test/01

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.[3]

The file '0510_217.CDF' is a GC-MS experiment exported from Agilent ChemStation (located in 'pyms-data'). This file can be loaded in the memory as follows:

```
>>> from pyms.IO.ANDI.Class import ChemStation
>>> andi_file = "/home/current/proj/PyMS/pyms-data/0510_217.CDF"
>>> andi_data = ChemStation(andi_file)
 -> Processing netCDF file '/home/current/proj/PyMS/pyms-data/0510_217.CDF'
    [ 2784 scans, masses from 50 to 550 ]
>>>
```

The above command creates the object 'andi_data' which is an *instance* of the class IO.ANDI.ChemStation.

2.2.2 Exploring an ANDI-MS data object

The object 'andi_data' has several attributes and methods associated with it.

```
>>> print "ANDI-MS data filename:", andi_data.get_filename()
ANDI-MS data filename: /home/current/proj/PyMS/pyms-data/0510_217.CDF
```

The method `get_tic()` return total ion chromatogram (TIC) of the data as an IonChromatogram object:

```
tic = andi_data.get_tic()
```

An IonChromatogram object is a one dimensional vector containing mass intensities as a function of retention time. This can can be either m/z channel intensities (for example, ion chromatograms at m/z = 65), or cumulative intensities over all measured m/z (TIC).

The method `get_ic_at_index(i)` returns i-th ion chromatogram, as an IonChromatogram object. For example, to get the first ion chromatogram from the data:

[3] 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.

```
ic = andi_data.get_ic_at_index(1)
```

The method `get_ic_at_mass(MZ)` returns the ion chromatogram for m/z = MZ. For example, to get the ion chromatogram that corresponds to m/z = 73:

An ion chromatogram object has a method `is_tic()` which returns True is the ion chromatogram is TIC, False otherwise:

```
>>> print "'tic' is a TIC:", tic.is_tic()
'tic' is a TIC: True
>>> print "'ic' is a TIC:",ic.is_tic()
'ic' is a TIC: False
```

2.2.3 Writing data to a file

The method `write()` of IonChromatogram object allows one to save the ion chromatogram object to a file:

```
>>> tic.write("output/tic.dat", minutes=True)
>>> ic.write("output/ic.dat", minutes=True)
```

The flag minutes=True indicates that retention time will be saved in minutes. The ion chromatogram object saved with with the `write` method is a plain ASCII file which contains a pair of (retention time, intensity) per line:

```
$ head tic.dat
  5.0944      745997.0000
  5.1002      726566.0000
  5.1059      717704.0000
  5.1116      684214.0000
  5.1173      701866.0000
  5.1230      893306.0000
  5.1287     1278099.0000
  5.1345     1290984.0000
  5.1402      925558.0000
  5.1459      644122.0000
```

Figure 2.1 shows the plot of the file 'tic.dat' produced with the program Gnuplot. The Gnuplot script used to produce this plot is provided as pyms-test/01/output/plot.gnu.

The method `get_intensity_matrix()` of ChemStation object returns the entire matrix of intensities:

```
>>> im = andi_data.get_intensity_matrix()
```

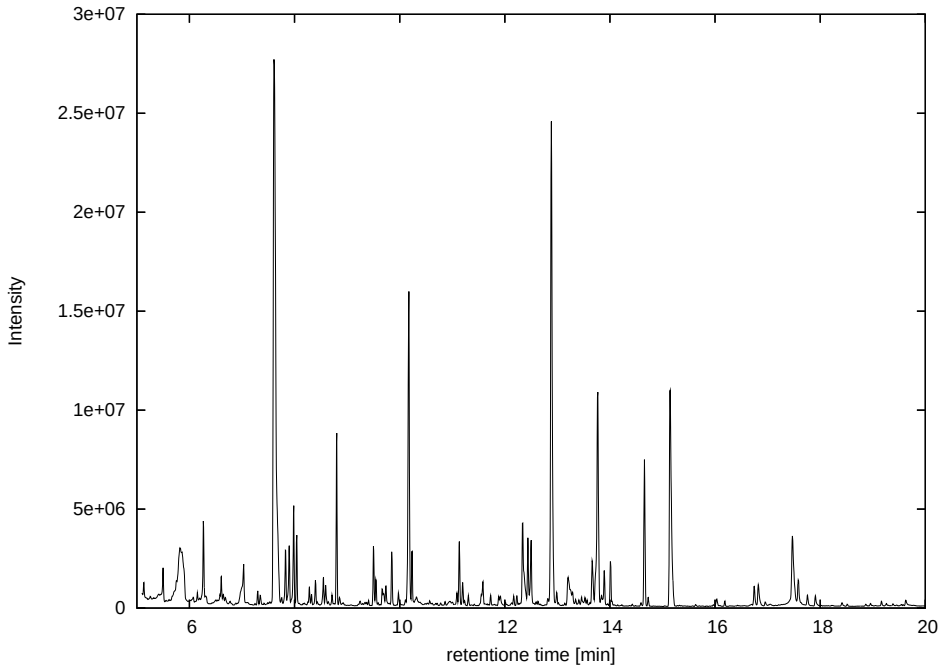

Figure 2.1: The Gnuplot plot of the file 'tic.dat'

```
>>> print "Dimensions of the intensity matrix are:",len(im),"x",len(im[0])
Dimensions of the intensity matrix are: 2784 x 501
```

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

The intensity matrix can be saved to a file with the function 'save_data()':

```
save_data("output/im.dat", im)
```

The entire data (ie. ChemStation object) can be saved as CSV with the method `export_csv()`. For example,

```
>>> andi_data.export_csv("output/data")
```

will create 'data.im.csv, data.mz.csv, and data.rt.csv where these are the intensity matrix, retention time vector, and m/z vector in the CSV format.

2.3 Example 2: Creating signal peaks

This example is in pyms-test/02

In PyMS a signal peak is represented as 'Peak' object defined in pyms.Peak.Class.py. A peak object is initialized with two arguments: peak retention time and peak raw area. The following commands create a peak named 'p' with the retetion time of 5.553 min and a peak area of 2759280 (this is the peak no. 3 in the ChemStation peak area report file 'a0806_140.txt'):

```
>>> from pyms.Peak.Class import Peak
>>> p = Peak(5.553*60.0,2759280)
```

As a matter of convention PyMS internally stores retention times in seconds, hence above the retention time is multiplied by 60. Peak raw area is in arbitrary units.

Peak properties can be accessed through its attributes:

```
>>> print "Peak retention time is", p.rt
Peak retention time is 333.18
>>> print "Peak raw area is", p.raw_area
Peak raw area is 2759280.0
```

Other important properties of a peak object are peak normalized area and peak mass spectrum. The peak created in the above example does not have values associated with these two attributes, and they are merely initialized to 'None':

```
>>> print "Peak normalized area is", p.norm_area
Peak normalized area is None
>>> print "Peak mass spectrum is", p.mass_spectrum
Peak mass spectrum is None
```

The peak mass spectrum can be set by calling the method `set_mass_spectrum()`. This method requires the raw data, and fetches mass spectrum at peak retention time:

```
>>> p.set_mass_spectrum(andi_data)
```

This will set the mass spectrum attribute:

```
>>> print p.mass_spectrum
 49976  54520 102752  15570   1872  18392   8765  14966  46136  16141
  1635   1743    686   1019    712   1199   1641   3182   1234  30400
  4261   3746   3348  82392   8354  24824   3797   6086  23312 140480
[--output deleted--]
```

These are m/z channel intensities in arbitrary units. The m/z values themselves are in the mass list attribute:

```
>>> print p.mass_list
[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81,
[--outout deleted--]
```

The length of the two arrays must match:

```
>>> print len(p.mass_spectrum)
501
>>> print len(p.mass_list)
501
```

The mass spectrum can be written to a file by calling the peak `write_mass_spectum()` method:

```
>>> p.write_mass_spectrum("output/ms.dat")
```

The file 'output/ms.dat' contains the pairs (mz, intensity), one pair per line:

```
$ head output/ms.dat
  50.000        49976.000
  51.000        54520.000
  52.000       102752.000
  53.000        15570.000
  54.000         1872.000
  55.000        18392.000
  56.000         8765.000
  57.000        14966.000
  58.000        46136.000
  59.000        16141.000
```

Figure 2.2 shows the plot of ms.dat created with the program Gnuplot. The gnuplot script used to create this plot is provided as pyms-test/02/output/plot.gnu.

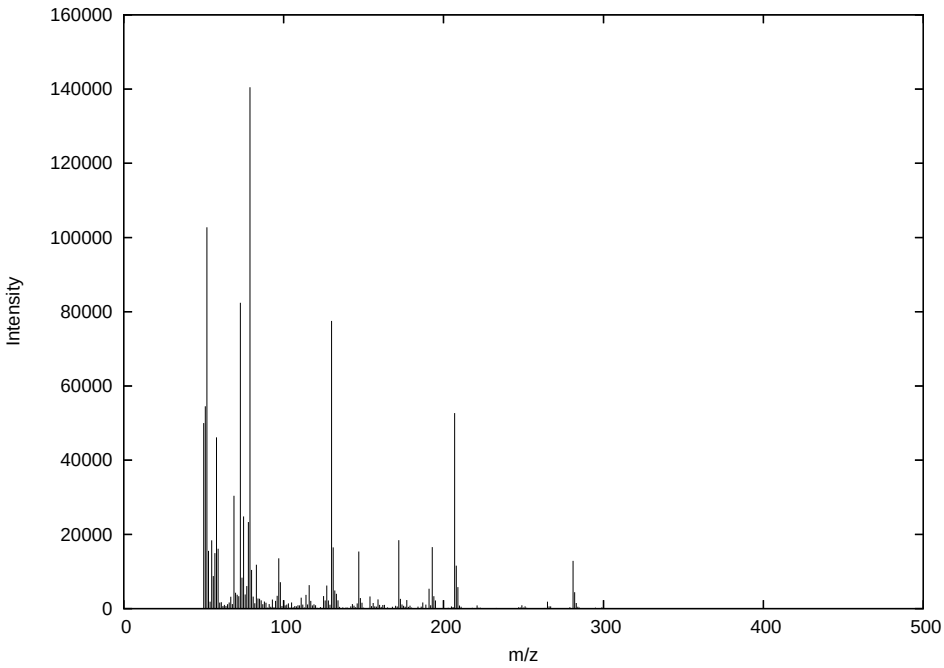

Figure 2.2: The plot of the file 'ms.dat' produced with Gnuplot

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.