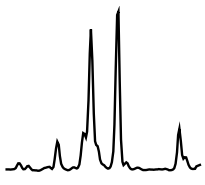

# PyMS User Guide



PyMS version 1.0

PyMS is a modular software package for processing of chromatography–mass spectrometry data with emphasis on scripting capabilities.

# Contents

# Introduction

## 1.1 Chromatography – mass spectrometry

PyMS is software for processing of chromatography–mass spectrometry data. Mass spectrometry is an analytical technique widely used in chemistry and biology for research, quality control studies, and forensic analysis. Mass spectrometry analysis is based on detection ions separated by their different mass-to-charge (m/z) ratio. The ions can be generated by a variety of methods (thermal ionization, chemical ionization, electron impact ionization, etc), and can be ionized atoms or molecular fragments. The separation by m/z ratio could also be achieved in a variety of ways (static or dynamic electric or magnetic fields). The combinations of ionization methods and ion separation methods leads to a great variety of mass spectrometers. As a consequence today it is possible to purchase many different mass spectrometer instruments which can be used in a variety of experimental setups. For more information see [1].

*Gas chromatography.* Chromatography separation relies on a column packed with a stationary phase (immobile) over which a mobile phase is pushed. The mobile phase is either gas (gas chromatography, GC) or liquid (liquid chromatography, LC). The mobile phase carries the complex mixture to be separated. The two phases are chosen so that components of the mixture have different solubilities in each phase. Each component of the mixture is in equilibrium between the stationary and the mobile phase, determined by its solubility in the two phases. As a result components of the mixture (different molecules) travel with different speed through the column, resulting in the separation of the molecular mixture.

*Mass spectrometry.* Mass spectrometry analyzer is located at the outlet of the chromatography column. As the mobile phase leaves the column (possibly with some component of the sample mixture) the mass spectrometer records a full mass spectrum (m/z vs intensity). Recording one such spectrum is called a "scan", and mass spectrometer operates in repetitive scanning mode. The separation occurs on the time scale of minutes, and a certain number of scans is performed per unit time to record mass spectra continuously.

*Hyphenated mass spectrometry data.* Joining chromatography separation with mass spectrometer results in hyphenated mass spectrometry, GC-MS or LC-MS, depending on the nature of the chromatoraphic separation. The time between the sample injection and the detection of an analyte on the detector at the end of the column is called the retention time. Retention time is unique for the analyte, i.e. each compound in a mixture. The data sets resulting from GC-MS and LC-MS are complex. The raw data

could be viewed as a two dimensional matrix, with the retention time along one dimension. For each time point a mass spectrum appears in the second dimension. Often the data is viewed as the projection along the time axis: for each time point all m/z values are summed to give one-dimensional spectrum (total ion chromatogram or TIC). Gas chromatography coupled to mass spectrometry (GC-MS) is suitable for profiling of volatile, thermally stable metabolites (or metabolites made such by chemical derivatization), while liquid chromatography coupled to mass spectrometry (LC-MS) is suitable for profiling of predominantly polar metabolites [2].

The principles of GC-MS and LC-MS data processing are well established. A typical data processing pipeline may involve noise attenuation, baseline correction, peak detection, and peak quantitaion (integration). Overlapping signals may be discerned by processing ion chromatograms for individual masses (deconvolution), or two-dimensional data processing methods may be applied directly on the spectral matrix. In GC-MS mass spectra for individual time points are often matched against large libraries of mass spectra for compound identification purposes.

## 1.2 Metabolite profiling (metabolomics)

Metabolite profiling is an emerging field of functional genomics, highly complementary to transcript profiling (transcriptomics) and protein profiling (proteomics), with significant promises for applications in biomarker discovery, toxicology, nutrition, basic metabolic research and integrated systems biology [3, 4]. Metabolomics refers to an all-inclusive profiling of low-molecular weight metabolites with an implicit aim to interpret the results in the context of the organism's genome and its global metabolic network. Hyphenated mass spectrometry and nuclear magnetic resonance (NMR) are the principal analytical techniques in metabolomics. Chromatography separation coupled to mass spectrometry detection is a robust analytical approach used to quantitate hundreds of compounds in metabolomic studies.

Non-targeted metabolite profiling has brought about new challenges in the development of data processing methods able to support large scale, high-throughput experiments. Data processing for high-throughput metabolomic experiments is still more a defining goal than reality, and may require utilization of emerging computing platforms such as distributed and Grid computing to speed up the processing. A related challenge in the domain of bioinformatics is the effective application of statistical, machine learning, and data mining methods required to extract useful information from data.

## 1.3 About PyMS

PyMS is a modular software package for processing of chromatography–mass spectrometry data with emphasis on scripting capabilities. The idea behind PyMS is to decouple processing methods form visualization and the concept of interactive processing altogether. The purpose of this is to provide a set of components for rapid development and testing of new processing methods and algorithms, as well as automated data processing.

PyMS is released as open source, under the GNU Public License version 2.

# Using PyMS

## 2.4 Running PyMS

PyMS can be run either interactively or from the script. When run in an interactive session commands are issued interactively through a Python shell. When run in the script mode the commands are collected into a file which is then passed to python for processing.

It is possible to run PyMS under the default Python shell. However we recommend using IPython, enhanced python shell [5], for interactive work with PyMS. IPython has a number of features that make interactive work easier.

## 2.5 PyMS components

PyMS is a Python package which contains a number of sub-packages, each of which provides certain functionality. This includes for example,

- Baseline/ – Baseline correction module.
- Display/ – Visualisation module.
- IO/ – Contains functions for reading of ANDI/MS data.
- Noise/ – Noise smoothing and analysis.
- Peak/ – Peak-related modules (including manipulation of peak lists).

PyMS is a collection of loosely coupled sub-packages. It is possible to load only one or a few sub-packages in Python (i.e. a part of PyMS), if a limited functionality is required. For example, it is possible to read experimental data without having other parts of PyMS available. Reading experimental data in the ANDI/MS format requires that the sub-package pyms.IO is loaded in the Python environment, and this can be done independently from loading other sub-packages, such as pyms.Baseline (provide baseline correction) or pyms.Display (provides visualisation).

## 2.6 Reading ANDI/MS data

ANDI/MS data format stands for Analytical Data Interchange for Mass Spectrometry, and is model for the description of mass spectrometric data developed in 1994 by Analytical Instrument Association. ANDI/MS is essentially a recommendation. 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 specification 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.

The sub-package pyms.IO provides capabilities to read the raw GC-MS data in the ANDI/MS format. The basic function to read ANDI/MS data provided by the pyms.IO subpackage is IO.ANDI.ChemStation(). The name is a reminder that this function has been reasonably tested only on the data exported from Agilent ChemStation software. In fact, IO.ANDI.ChemStation() was implemented based on comparisons with ChemStation, i.e. data read by PyMS was compared to the the original data in ChemStation to confirm that IO.ANDI.ChemStation() reads the data properly.

In theory the same function should be able to read files created by other software packages that supports export to ANDI/MS. Quick tests suggests that IO.ANDI.ChemStation() can read properly ANDI/MS exported from at least few other vendors, including Leco (ChromaTOF software). This however requires a proper testing and verification, because of the reasons mentioned before.

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

```
In [1]: from pyms import IO

In [2]: data = IO.ANDI.ChemStation("CHEMSTATION.CDF")
 -> Processing netCDF file 'CHEMSTATION.CDF'
    [ 2784 scans, masses from 50 to 550 ]
```

Where 'CHEMSTATION.CDF' is the GC-MS data in ANDI/MS format.

At this point it would be helpful to introduce a few terms. IO.ANDI.ChemStation() is actually a *class*, and the above command creates the object 'data' which is an *instance* of the class IO.ANDI.ChemStation. A class is a template for making instances. A class may have one or more *attributes* which are inherited by every instance made out of the class, and initialized when the instance is created.

The above command creates the object `data`, which has the following methods associated with is:

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

  ```
  In [3]: data.get_filename()
  Out[3]: 'CHEMSTATION.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:

```
In [3]: ic = data.get_ic_at_index(1)
```

- `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:

  ```
  In [3]: ic = data.get_ic_at_mass(73)
  ```

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

  ```
  In [3]: im = data.get_intensity_matrix()
  In [4]: len(im)
  Out[4]: 2784
  In [5]: len(im[0])
  Out[5]: 501
  ```

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

# Algorithms

## 3.7 Minmax peak detector

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 [6] and XCMS [7]. MZmine detects peaks by finding local maxima of a certain width [6]. In XCMS peaks are detected by using an empirical signal-to-noise cutoff after matched filtration with a second-derivative Gaussian [7]. 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.[1]

   (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

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

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 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: $M = 3$, $Q = 1.0°$).

# PyMS Installation

The instructions below refer to installation under Linux, and assume a reasonably savvy Linux user. Python is however fully cross-platform compatible, and PyMS should work equally well under other operating systems as long as the dependencies can be satisfied.

## 4.8 Installing core dependencies

### 4.8.1 Preparing the environment for a local installation

There are two methods of installing PyMS dependencies. The first is *system-wide*, when root access is available. The second is *locally*, when write access to the system directories is unavailable, or when PyMS is only used by one user. In this case, the dependencies are installed into a user's home directory.

Before performing a local installation, the following extra two steps are required.

**Directory structure:** Create a directory for the dependencies to install into. For this guide, we choose ~/pyms-deps.

**Environment variables:** The $PATH variable needs to be configured for the target installation directory. In the case of the bash shell, add `PATH=$HOME/pyms-deps/bin:$PATH` to your .bash_profile file, then reopen the terminal.

### 4.8.2 Python

Python is a general purpose object oriented programming language, which PyMS is written in. Python is open source and freely available from `http://www.python.org/download`.

The instructions below refer to the installation of Python version 2.5.

**Filename:** `Python-2.5.tgz`

**Installation:**

1. `tar -zxvf Python-2.5.tgz`
2. `cd Python-2.5`
3. `./configure` *if installing system-wide, or*
   `./configure --prefix=$HOME/pyms-deps` *if installing locally*
4. `make`
5. `make install`

Please refer to the file `Python-2.5/README` for more details.

### 4.8.3 netCDF

netCDF (network Common Data Form) is an interface for array-oriented data access and a library that provides an implementation of the interface. The netCDF library also defines a machine-independent format for representing scientific data. Together, the interface, library, and format support the creation, access, and sharing of scientific data.

netCDF is available from `http://www.unidata.ucar.edu/downloads/netcdf/`.

The description below refers to the installation of netCDF version 3.6.1.

**Filename:** `netcdf-3.6.1.tar.gz`

**Installation:**

1. `tar -zxvf netcdf-3.6.1.tar.gz`
2. `cd netcdf-3.6.1/src`
3. `./configure` *if installing system-wide, or*
   `./configure --prefix=$HOME/pyms-deps` *if installing locally*
4. `make`
5. `make install`

Please refer to the file `netcdf-3.6.1/src/INSTALL` for more details.

### 4.8.4 numarray

numarray is a set of extensions to the Python programming language which allows large sets of numerical values to be efficiently manipulated.

numarray is available from `http://www.stsci.edu/resources/software_hardware/numarray`.

The description below refers to the installation of numarray version 1.5.2.

**Filename:** `numarray-1.5.2.tar.gz`

**Installation:**

1. `tar -zxvf numarray-1.5.2.tar.gz`
2. `cd numarray-1.5.2`
3. `python setup.py config install --gencode`

Please refer to the file `numarray-1.5.2/Doc/INSTALL.txt` for more details.

### 4.8.5 pycdf

pycdf is a set of extensions to the Python programming language providing an interface to to the netCDF library. The description below refers to the installation of pycdf version 0.6-2-rc1.

pycdf is available from `http://pysclint.sourceforge.net/pycdf/`.

The description below refers to the installation of pycdf version 0.6-2-rc1.

**Filename:** `pycdf-0.6-2-rc1.tar.gz`

**Installation:**

1. `tar -zxvf pycdf-0.6-2-rc1.tar.gz`
2. `cd pycdf-0.6-2-rc1`
3. `vi setup.py`
   (a) At line 31 (`USE = NUMERIC`), add a # at the start of the line
   (b) At line 32 (`#USE = NUMARRAY`), remove the # at the start of the line
   (c) Save the file and exit
4. `python setup.py install`

Please refer to the file `pycdf-0.6-2-rc1/INSTALL` for more details.

### 4.8.6 matplotlib

matplotlib is a python 2D plotting library. It is used by pyms for visualizations of data.

matplotlib is available from `http://matplotlib.sourceforge.net/`.

The description below refers to the installation of matplotlib version 0.87.7.

**Filename:** `matplotlib-0.87.7.tar.gz`

**Installation:**

1. Ensure that you have the preequisites for matplotlib's GUI extensions, as the plots will not display otherwise. Any one of the following can be used:

   (a) pygtk
   (b) wxpython
   (c) tk
2. `tar -zxvf matplotlib-0.87.7.tar.gz`
3. `cd matplotlib-0.87.7`
4. `python setup.py build`
5. `python setup.py install`

Please refer to the file `matplotlib-0.87.7/INSTALL` for more details.

## 4.9 Installing optional dependencies

Optional dependencies are packages which are not required for the core functionality of PyMS, but may be required for some extra functionality, or are highly recommended (such as IPython, interactive Python shell).

### 4.9.1 IPython

IPython is enhanced Python shell suitable for interactive work.

IPython is available from `http://ipython.scipy.org/moin/`.

The description below refers to the installation of IPython version 0.7.3.

**Filename:** `ipython-0.7.3.tar.gz`

**Installation:**

1. `tar -zxvf ipython-0.7.3.tar.gz`
2. `cd ipython-0.7.3`
3. `python setup.py install`

Please refer to the file `ipython-0.7.3/doc/manual.pdf` for more details.

### 4.9.2 Numeric

Numeric is Python module for high-performance numeric computing. This module is required by Pycluster, and needs to be installed only if Pycluster is used (see below).

Numeric is available from `http://optusnet.dl.sourceforge.net/sourceforge/numpy/Numeric-24.2.tar.gz`.

The description below refers to the installation of Numeric version 24.2.

**Filename:** `Numeric-24.2.tar.gz`

**Installation:**

1. `tar -zxvf Numeric-24.2.tar.gz`
2. `cd Numeric-24.2`
3. `python setup.py build`
4. `python setup.py install`

Please refer to the file `Numeric-24.2/README` for more details.

### 4.9.3 Pycluster

Pycluster is a Python extension module to the clustering routines in the C Clustering Library by Michiel de Hoon. Most people will not use this PyMS, and the installation is highly optional.

Pycluster is available from `http://bonsai.ims.u-tokyo.ac.jp/~mdehoon/software/cluster/software.htm`.

The description below refers to the installation of Pycluster version 1.33.

**Filename:** `Pycluster-1.33.tar.gz`

**Installation:**

1. `tar -zxvf Pycluster-1.33.tar.gz`
2. `cd Pycluster-1.33`
3. `python setup.py install`

## 4.10 PyMS

### 4.10.1 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 servers maintain 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[2], for more information see

[2]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'.

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 in the current directory:

```
$ 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
.....
A    pyms/Noise/Window.py
Checked out revision 30.
$ ls -CF
pyms/
$
```

### 4.10.2 PyMS installation

The process described above shows the installation of netCDF, numarray, and pycdf within the Python standard distribution. This is typically a directory named "site-packages" withini the Python installation directory (for example, /usr/local/lib/python2.4). For example, if "pycdf" and "numarray" are installed listing files in this directory would show:

```
$ ls -CF /usr/local/lib/python2.4/site-packages
pycdf/ numarray/
$
```

To install PyMS one needs to copy PyMS source directory to the site-packages directory:

```
$ cp -r pyms /usr/local/lib/python2.4/site-packages
```

The easiest way to test if PyMS has been installed properly is to attempt to import the package within the Python environment:

```
$ python
Python 2.4.1 (#1, Jun 27 2005, 12:53:02)
[GCC 3.4.3 20041212 (Red Hat 3.4.3-9.EL4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyms
>>>
```

It there are no feedback messaged to the import command, Python was able to load PyMS.

### 4.10.3 Installing PyMS elsewhere

Installing PyMS in the python standard directory may require super-user password. The alternative installation is to install PyMS somewhere locally. For example, the directory pyms/ with PyMS source code could be placed in the directory /home/projects/. To make Python aware of PyMS the following commands are required before any part of PyMS is called:

```
import sys
sys.path.append("/home/projects")
```

Alternatively the environment variable PYTHONPATH could be set to ”/home/projects/”. For example, under the bash shell:

```
$ PYTHONPATH=/home/projects/; export PYTHONPATH
```

More information about sys.path.append() and PYTHONPATH is possible to find in Python documentation.

## 4.11 IPython

IPython is a substitute for the stock Python interactive shell. It provides several features which enhances the day-to-day workflow of using PyMS, but is not required for its operation.

Its improvements include:

1. **Tab completion:** Press the `TAB key` to autocomplete variable and function names, and directory entries.
2. **Object inspection:** Type `$objectname` to obtain information and documentation about it.
3. **Shell passthrough:** Type `!commandname` to execute a shell command. The output of the command can be assigned to a variable if necessary.
4. **Color highlighting:** Input and output are highlighted in different colors for easier distinguishing.
5. **Command history:** Press `UP` or `DOWN` to scroll through the command history, and `CTRL-R` to search through the history. The history is persistent across sessions.
6. **Logging:** Input can be logged to a file, allowing for future inspection or replay. Type `%logstart logname` in IPython to activate logging.

IPython can be obtained from `http://ipython.scipy.org/dist/`. The description below refers to the installation of IPython version 0.7.2.

**Filename:** `ipython-0.7.2.tar.gz` (or newer)

**Installation:**

1. `tar -zxvf ipython-0.7.2.tar.gz`
2. `cd ipython-0.7.2`
3. `python setup.py build`
4. `python setup.py install`

# Bibliography


[1] Gross JH. *Mass Spectrometry*. Springer, Berlin, 2004. ISBN 3-540-40739-1.

[2] Halket JM, Waterman D, Przyborowska AM, Patel RK, Fraser PD, and Bramley PM. Chemical derivatization and mass spectral libraries in metabolic profiling by GC/MS and LC/MS/MS. *J Exp Bot*, 6(410):219–43, 2005.

[3] Fiehn O, Kopka J, Dormann P, Altmann T, Trethewey RN, and Willmitzer L. Metabolite profiling for plant functional genomics. *Nature biotechnology*, 18(11):1142–1143, 2000.

[4] Allen J, Davey HM, Broadhurst D, Heald JK, Rowland JJ, Oliver SG, and Kell DB. High-throughput classification of yeast mutants for functional genomics using metabolic footprinting. *Nature biotechnology*, 21(6):692–696, 2003.

[5] IPython. http://ipython.scipy.org/moin/.

[6] 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.

[7] 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.