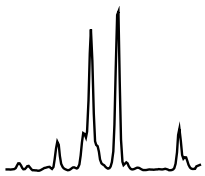

PyMS2 User Guide

PyMS version 1.0

A Python toolkit for processing of chromatography–mass spectrometry data

Contents

Introduction **1**

1.1 About PyMS . . . 1

1.2 PyMS installation . . . 1

1.2.1 Downloading PyMS source code . . . 2

1.2.2 PyMS installation . . . 2

1.2.3 Package 'NumPy' . . . 3

1.2.4 Package 'pycdf' (required for reading ANDI-MS files) . . . 3

1.2.5 Package 'Pycluster' (required for peak alignment by dynamic programming) . . . 3

1.2.6 Package 'scipy.ndimage' (required for TopHat baseline correction) . . . 3

1.3 Current PyMS development environment . . . 3

1.4 Troubleshootings . . . 5

1.4.1 Pycdf import error . . . 5

1.5 PyMS tutorial and examples . . . 1

Fundamental data structures **3**

2.1 GC-MS Data Model . . . 3

2.1.1 Reading JCAMP GC-MS data into PyMS . . . 3

2.1.2 Reading ANDI GC-MS data into PyMS . . . 4

2.1.3 Exploring a GCMS data object . . . 4

2.1.4 Exploring a Scan data object . . . 5

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

Each part is a separate project on Google Code that can be downloaded separately. The data used in PyMS documentation and examples is available from the Bio21 Institute server:
`http://bioinformatics.bio21.unimelb.edu.au/pyms-data/`
In addition, the current PyMS API documentation is available from here:
`http://bioinformatics.bio21.unimelb.edu.au/pyms.api/index.html`

1.2 PyMS installation

There are several 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 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 with the program '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.
```

1.2.2 PyMS installation

PyMS installation consists of placing the PyMS code directory (pyms/) in place visible to Python interpreter. This can be in the standard place for 3rd party software (the directory site-packages/). If PyMS code is placed in a non-standard place the Python interpreter needs to be made aware of it before before it is possible to import PyMS modules (see the Python sys.path.append() command).

We recommend compiling your own Python installation for PyMS.

In addtion to the PyMS core source code, a number of external packages is used to provide additional functionality. These are explained below.

[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'.

1.2.3 Package 'NumPy'

The package NumPy is provides numerical capabilities to Python. This package is used throughout PyMS (and also required for some external packages used in PyMS), to its installation is mandatory.

The NumPy web site `http://numpy.scipy.org/` provides the installation instructions and the link to the source code.

1.2.4 Package 'pycdf' (required for reading ANDI-MS files)

The pycdf (a python interface to Unidata netCDF library) source and installation instructions can be downloaded from `http://pysclint.sourceforge.net/pycdf/`. Follow the installation instructions to install pycdf.

1.2.5 Package 'Pycluster' (required for peak alignment by dynamic programming)

The peak alignment by dynamic programming is located in the subpackage pyms.Peak.List.DPA. This subpackage used the Python package 'Pycluster' as the clustering engine. Pycluster with its installation instructions can be found here: `http://bonsai.ims.u-tokyo.ac.jp/ mdehoon/software/cluster/index.html`.

1.2.6 Package 'scipy.ndimage' (required for TopHat baseline correction)

If the full SciPy package is installed the 'ndimage' will be available. However the SciPy contains large amount of functionality, and its intallation is somewhat involved. In some situations in may be preferable to install only the subpackage 'ndimage'. The UrbanSim web site [2] provides instructions how to install a local copy of 'ndimage'. These instructions and the link to the file 'ndimage.zip' are here:
`http://www.urbansim.org/opus/releases/opus-4-1-1/docs/installation/scipy.html`

1.3 Current PyMS development environment

PyMS is currently being developed with the following packages:

```
Python-2.5.2
numpy-1.1.1
netcdf-4.0
pycdf-0.6-3b
Pycluster-1.41
```

A quick installation guide for packages required by PyMS is given below.

1. Python installation:

```
$ tar xvfz Python-2.5.2.tgz
$ cd Python-2.5.2
$ ./configure
$ make
$ make install
```

   This installs python in /usr/local/lib/python2.5. Make sure that python called from the command line is the one just compiled and installed.

2. NumPy installation:

```
$ tar xvfz numpy-1.1.1.tar.gz
$ cd numpy-1.1.1
$ python setup.py install
```

3. pycdf installation

   Pycdf has two dependencies: the Unidata netcdf library and NumPy. The NumPy installation is described above. To install pycdf, the netcdf library must be downloaded (`http://www.unidata.ucar.edu/software/`
   compiled and istalled first:

```
$ tar xvfz netcdf.tar.gz
$ cd netcdf-4.0
$ ./configure
$ make
$ make install
```

   The last step will create several binary 'libnetcdf*' files in /usr/local/lib. pycdf can be installed as follows:

```
$ tar xvfz pycdf-0.6-3b
$ cd pycdf-0.6-3b
$ python setup.py install
```

4. Pycluster installation

```
$ tar xvfz Pycluster-1.42.tar.gz
$ cd Pycluster-1.42
$ python setup.py install
```

5. ndimage installation:

```
$ unzip ndimage.zip
$ cd ndimage
$ python setup.py install --prefix=/usr/local
```

Since ndimage was installed outside the scipy package, this requires some manual correction:

```
$ cd /usr/local/lib/python2.5/site-packages
$ mkdir scipy
$ touch scipy/__init__.py
$ mv ndimage scipy
```

1.4 Troubleshootings

The PyMS is essentially a python library (a 'package' in python parlance, which consists of several 'sub-packages'), which for some functionality depends on other python libraries, such as NumPy, pycdf, and Pycluster. The most likely problem with PyMS installation is a problem with installing one of the PyMS dependencies.

1.4.1 Pycdf import error

On Red Hat Linux 5 the SELinux is enabled by default, and this causes the following error while trying to import properly installed pycdf:

```
$ python
Python 2.5.2 (r252:60911, Nov  5 2008, 16:25:39)
[GCC 4.1.1 20070105 (Red Hat 4.1.1-52)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pycdf
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.5/site-packages/pycdf/__init__.py", line 22, in <module>
    from pycdf import *
  File "/usr/local/lib/python2.5/site-packages/pycdf/pycdf.py", line 1096, in <module>
    import pycdfext as _C
  File "/usr/local/lib/python2.5/site-packages/pycdf/pycdfext.py", line 5, in <module>
    import _pycdfext
ImportError: /usr/local/lib/python2.5/site-packages/pycdf/_pycdfext.so:
    cannot restore segment prot after reloc: Permission denied
```

This problem is removed simply by disabling SELinux (login as 'root', open the menu Administration → Security Level and Firewall, tab SELinux, change settings from 'Enforcing' to 'Disabled').

This problem is likely to occur on Red Hat Linux derivative distributions such as CentOS.

1.5 PyMS tutorial and examples

A tutorial illustrating various PyMS features in provided in subsequent chapter of this User Guide. The commands executed interactively are grouped together by example, and provided as Python scripts in the project 'pyms-test' (this is a Google code project, similar to the project 'pyms' which contains the PyMS source code).

The setup used in the examples below is as follows. The projects 'pyms', 'pyms-test', 'pyms-docs', and 'data' are all in the same directory, '/x/PyMS'. In the project 'pyms-test' there is a directory corresponding to each example coded with the example number (ie. `pyms-test/21a/` corresponds to Example 1a in Chapter 2).

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 with the following command:

```
$ 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("/x/PyMS/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 may need to be adjusted to match your own directory structure.

All data files (raw data files, peak lists etc.) used in the example below can be found at `http://bioinformatics.bio21.un` and are assumed to be located in the 'data' directory.

Fundamental data structures

2.1 GC-MS Data Model

PyMS can read gas chromatography-mass spectrometry (GC-MS) data stored in Analytical Data Interchange for Mass Spectrometry (ANDI-MS),[2] and Joint Committee on Atomic and Molecular Physical Data (JCAMP-DX)[3] formats. These formats are essentially recommendations, and it is up to individual vendors of mass spectrometry processing software to implement "export to ANDI-MS" or "export to JCAMP-DX" features in their software. It is also possible to get third party converters. The information contained in the exported data files can vary significantly, depending on the instrument, vendor's software, or conversion utility.

For PyMS, the minimum set of assumptions about the information contained in the data file are:

- The data contain the mz and intensity value pairs across a scan.
- Each scan has a retention time.

Internally, PyMS stores the raw data from ANDI files or JCAMP files as a GCMS_data object. Once a GCMS_data object has been created, the raw data can not be modified.

2.1.1 Reading JCAMP GC-MS data into PyMS

[*This example is in pyms-test/20a*]

The PyMS package pyms.GCMS.IO.JCAMP provides capabilities to read the raw GC-MS data stored in the JCAMP-DX format.

The file 'gc01_0812_066.jdx' (located in 'data') is a GC-MS experiment converted from Agilent ChemStation format to JCAMP format using File Translator Pro.[4] This file can be loaded in Python as follows:

[2] ANDI-MS was developed by the Analytical Instrument Association.
[3] JCAMP-DX is maintained by the International Union of Pure and Applied Chemistry.
[4] ChemSW, Inc.

```
>>> from pyms.GCMS.IO.JCAMP.Function import JCAMP_reader
>>> jcamp_file = "/x/PyMS/data/gc01_0812_066.jdx"
>>> data = JCAMP_reader(jcamp_file)
 -> Reading JCAMP file '/x/PyMS/pyms-data/gc01_0812_066.jdx'
>>>
```

The above command creates the object 'data' which is an *instance* of the class GCMS_data.

2.1.2 Reading ANDI GC-MS data into PyMS

[*This example is in pyms-test/20b*]

The PyMS package pyms.GCMS.IO.ANDI provides capabilities to read the raw GC-MS data stored in the ANDI-MS format.

The file 'gc01_0812_066.cdf' (located in 'data') is a GC-MS experiment converted to ANDI-MS format from Agilent ChemStation (from the same data as in example 01a above). This file can be loaded as follows:

```
>>> from pyms.GCMS.IO.ANDI.Functions import ANDI_reader
>>> ANDI_file = "/x/PyMS/data/gc01_0812_066.cdf"
>>> data = ANDI_reader(ANDI_file)
 -> Reading netCDF file '/x/PyMS/pyms-data/gc01_0812_066.cdf'
>>>
```

The above command creates the object 'data' which is an *instance* of the class GCMS_data.

2.1.3 Exploring a GCMS data object

[*The following examples are the same in pyms-test/20a and pyms-test/20b*]

The object 'data' (from the two previous examples) stores the raw data as a *GCMS_data* object. Within the GCMS_data object, raw data are stored as a list of *Scan* objects and a list of retention times. There are several methods available to access data and attributes of the GCMS_data and Scan objects.

The GCMS_data object's methods relate to the raw data. The main properties relate to the masses, retention times and scans. For example, the minimum and maximum mass from all of the raw data can be returned by the following:

```
>>> data.get_min_mass()
>>> data.get_max_mass()
```

A list of all retention times can be returned by:

```
>>> time = data.get_time_list()
```

The index of a specific retention time (in seconds) can returned by:

```
>>> data.get_index_at_time(400.0)
```

Note that this returns the index of the retention time in the data closest to the given retention time (400.0 seconds).

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

```
tic = data.get_tic()
```

The IonChromatogram object is covered in a later section.

A list of all the raw Scan objects can be returned by:

```
>>> scans = data.get_scan_list()
```

The Scan object has methods relating to an individual scan. This is covered in the next section.

2.1.4 Exploring a Scan data object

[*The following examples are the same in pyms-test/01a and pyms-test/01b*]

A Scan object contains a list of masses and a corresponding list of intensity values from a single mass-spectrum scan.

A list of all masses in a scan (e.g. the 1st scan) is returned by:

```
>>> scans[0].get_mass_list()
```

A list of all corresponding intensities in a scan is returned by:

```
>>> scans[0].get_intensity_list()
```

The maximum and minimum mass in an individual scan (e.g. the 1st scan) are returned by:

```
>>> scans[0].get_min_mass()
>>> scans[0].get_max_mass()
```

Bibliography

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

[2] UrbanSim. http://www.urbansim.org/.