PyMS Workshop

August/September 2010

1.1 Introduction

PyMS is a set of libraries for use in the processing of GCMS data. It is written in the Python programming language. The PyMS libaries are accessed through the use of scripts, and we will examine several scripts today. Though there are many ways such a library can be used, it is through scripting that we can make the most use of PyMS.

WebMS, a web based interface to PyMS is currently under development, but the versatility that a script can provide will never be matched by any graphical interface. The "Power User" of PyMS will always need to turn to scripting every now and then.

Python is an interpreted language, meaning that commands are executed in order as they are typed or passed to the Python interpreter. This is in contrast to many other languages where commands are bundled together, compiled into machine language and run.

This and several other factors make the learning curve for Python a lot less steep than for most other popular languages such as C++ or Java. Hopefully you will see that a little Python scripting is nothing to be afraid of, and certainly the modification of existing scripts is within the grasp of anyone at M.A.

1.2 Linux Basics

To interact with the Linux operating system we generally use the command line. For many applications you may also use the graphical user interface (GUI), but for PyMS and most scientific software in the Linux world the command line is essential. The main benefit of the command line approach is that commands are clear and precise and generally do exactly what you need them to do. From the developer's point of view command line driven software takes much less time to write. All energies are directed towards algorithm development, and the end product is a more transparent, easy to read and redevelop code.

Of course the main drawback is that you must learn the commands necessary to run any program you wish to run, but this is quickly learned once you are immersed in the Linux environment.

We will concentrate today on a few simple commands, and we will use the "Kate" text editor which has an easy to use and efficient GUI.

1.2.1 The console

The console is where you enter commands. To open a console, right click anywhere on the desktop and select the console option. A window with a blinking cursor should appear on screen. Any commands listed below in this tutorial should be entered in this screen.

1.2.2 The filesystem

The linux filesystem is almost always identical across the many linux flavours. To see the directory from which you are currently working use the "print working directory" command (type `pwd`).

The base of the filesystem is the `/` directory, and `/`'s you see after the leftmost one indicate levels of folders. For example, the folder `workshop/` where we will be working from today is located in the folder `/x/PyMS/` meaning that it is in the folder `PyMS/` which in turn is located in the folder `x/`, which is at the base of the filesystem `/`.

We need to navigate to the directory where our pre-written scripts are located. This is the `/x/PyMS/workshop/` folder on your filesystem. To change your working directory use the `cd` (change directory) command.

```
$ cd /x/PyMS/workshop/
```

Now if you type `ls`, you will see the contents of the directory.

These 3 commands, `pwd, cd,` and `ls` are all we require today, except for the running of scripts which we will come to later.

1.2.3 The gedit text editor

There are several simple good quality text editors available for Linux, but we will concentrate on one: gedit. gedit is quite easy to use, being similar to something like notepad in Windows. However it has some advanced capabilities which can be very useful when writing scripts.

One of these is it's ability to highlight syntax in scripts. Once gedit is aware that a file has a suffix such as .py (the python suffix), it can change how different parts of the file are displayed to make it easier to read and quickly scan. This is a huge benefit and it is advisable to use such a text editor whenever you need to modify a script.

To open the gedit editor type in a command window:

```
$ gedit&
```

The & at the end of the command puts the process in the background so that you can continue to type on the command line. Without the & you can no longer type on that command line without shutting down the process.

1.3 Scripting

A Python script has several parts. Let's take a look at one such script in detail.

```
"""proc-one-tic.py

Python script for MA workshop on PyMS, 19-08-10

This script plots a single TIC using the pyms Display module

"""

import sys
sys.path.append("/x/PyMS/")


from pyms.Display.Function import plot_ic
from pyms.Utils.IO import load_object


# Set the name of the folder where the file is
out_folder = "/home/socall/platform_compar/workshop/output/"

# Experiment label
# The name of the experiment
# This is the only line which needs to be changed
# to view the TIC of a different experiment
exper_name = "APC_BSTFA1_1"

# Load the TIC from the dumped IC object
tic = load_object(out_folder + exper_name + ".tic")

# Plot using the simple plotting function
# The second two arguments are optional, used to label the figure
plot_ic(tic, line_label = "TIC", plot_title = "TIC for " + exper_name)
```

This script is located in your current directory, so you can open it simply by clicking on the open file icon in gedit and selecting it.

```
$ python proc-one-tic.py
```

The first section is between the """. This is a comment, meaning that it is completely ignored by the python interpreter. You can use this space to detail the function of the script. Another way to add comments to the script is to place a "#" before the text in any line. The Python interpreter then skips that line completely. You can see that there are relatively few lines of actual code in this script, most are comments.

The second section imports the functions necessary for the running of the script. In this case we need to import "sys", so that we can change the working directory to `"/x/PyMS/"`, where the PyMS library is installed on the filesystem. Then we import the PyMS functions we need to run the script. The first of these is `load_object()`, which enables us to import the Total Ion Chromatogram (TIC) from an experiment. The second import from PyMS is the plot_ic function, which will allow us to plot the TIC.

Finally we use four commands to call the functions we want to use.

You can run this script from the command line by typing

```
$ python /x/PyMS/workshop/proc-plot-one-tic.py
```

You could perform the same function as the script simply by typing each line of code into the Python interpreter one by one - but I wouldn't recommend it.

1.4 Modifying existing scripts

You will see from the comments on the script above, that only one line of the script, in fact only one character on one line of the script, need be changed to look at a different TIC.

To examine a list of the different TIC's we may look at, type:

```
$ ls output/*.tic
```

Now change the line in the script to correspond to the experiment you are interested in to take a look at it's TIC.

Often a small one line or word change to a script is all that is needed to suit your requirements, and many scripts are available to perform the various tasks for which PyMS was designed.

1.5 A larger more complete script

We will now examine a typical script for PyMS which is capable of reading data files, filtering the data, and finding the peaks in the data. The script is called "proc-expr.py", and is also located in the `/x/PyMS/workshop/` directory. Open it in a text editor such as kate and follow along.

1.5.1 Importing and processing data files

PyMS can import data files in either the JCAMP or netCDF format, though it is more usual to use the netCDF format.

The function which does the import is called `ANDI_reader()`. As before, this needs to be imported into the script before it can later be called. To import it we use: `from pyms.GCMS.IO.ANDI.Function import ANDI_reader`

and to use it we type: `data = ANDI_reader("filename.CDF")`

where `data` is the resulting object into which PyMS has saved the data.

`data` now contains a matrix of intensities at many fractional masses, and we need to "bin" the intensity values to obtain a meaningful and useful picture of the experiment. Binning attributes neighbouring fractional masses to the nearest integer mass. To bin the data we use the function `build_intensity_matrix_i()`.

In actual fact the area of binning is a little more complex than I have detailed here, so for more information please consult the PyMS user guide, section 3.1.1.

1.5.2 Filtering

Each column of the intenstity matrix is a list of scans at a single m/z value - what we call an Ion Chromatogram. We do our filtering on the Intenstity Matrix object, correcting for both baseline and noise issues within the following code:

```
# smooth data
im_smooth = savitzky_golay(im)
im_base = tophat(im_smooth, struct="1.5m")
im = im_base # copy im_smooth to im
```

Filtering is in some ways a complicated science and in others a black art. We will not concentrate on the finer points. Essentially the same filtering technique can be applied to any GC-MS dataset without any modification.

1.5.3 Peak Finding

The peaks are found by examining the individual Ion Chromatograms (ICs) for local maxima across the time domain of the experiment. Where some maxima occur at the same time in different IC's we call this a peak. Hopefully a peak is a set of ions from a single compound eluting from the GC column at that time.

Several parameters are necessary for peak finding. The first is the size of the window over which you wish to look for local maxima - here called "points". We have found that a good rule of thumb is a half to a third of the number of "points per peak". By points per peak we mean the number of scans between the beginning and the end of a typical peak in the dataset.

The next parameter is "scans". We can have the problem of several ions from the same compound being detected in neighbouring scans. This is usually due to the scanning operation of the ion detecter which scans from high to low masses over the sampling period. Large masses eluting at the end of the sampling period will not be detected until the next scan. Usually we set "scans" to 2 so that any peaks detected in neighbouring scans are amalgamated into a single peak.

The function which does the peak detection is called `BillerBiemann()`, after the authors of the first paper on this algorithm. It was published in 1974 so is well established. We import it as before (have a look and see if you can find where we do this), then we invoke it with: `pl = BillerBiemann(im, points, scans)`.

The resulting list of peaks "pl" contains many peaks which do not make physical sense and we can filter these using some further parameters. We can use the fact that we expect several ions to be detected on the same scan for a peak to register and not just one or two. The parameter "`n`" determines this number and we usually set it to 3. We also set a threshold for this minimum number of ions "`t`", so that we must have "`n`" ions with an intensity of more than "`t`" in order for a peak in the peak list pl not to be discarded.

One last parameter acts on the peaks themselves rather than filtering the peak list. This parameter `r` is a percentage. Any ions within the peak with intensity less than `r%` of the highest intensity ion in the peak are set to zero.

We set all of these parameters with a simple `n = 3` type assignment, and you can easily see where this has been done in the script. Filtering the peak list is done with the functions `rel_threshold()` and `num_ions_threshold()`. Their use is best illustrated by referring to the script.

In my experience the parameter which must be changed most often is the threshold parameter. In fact we have made some efforts to automate this based on a measure of the median absolute deviation of the signal. We won't cover that today but you can find some information on it in the PyMS user guide, and I am available at any time for discussion of it.

The parameter "points" does change as your sampling period becomes shorter. For example we were using a window of 5 points when our sampling rate was 2.7/sec, but now use 13, with a sampling rate of 9/sec.

If you can understand what these parameters mean you will find it quite easy to operate the PyMS peak finding functions.

1.6 Visualising the peak list

The PyMS Display module is capable of displaying Ion Chromatograms, the TIC, and also the peak list. We have already seen earlier on the use of the `plot_ic()` function to display a single TIC.

We can use a more advanced plotting function, `plot_ics()` to display the TICs from all three experiments at once. The script to perform this is called "proc-all-tics.py". Here we use a loop to create a list of TICs, and send this list to the method `plot_ics()`. There are more advanced coding techniques in use here, but we will not go into them. Suffice to say that a slight modification to this script can make it useful for many plotting applications you might come across.

To display all of the ICs, the TIC, and the peaks on the same plot, we can use the script "plot-display-all.py". The resulting figure is shown below

1.7 Peak Alignment

Having now found peak lists for six different experiments, we turn to yet another script to align the peak lists. this script is called proc-align.py, and is also in your current working directory `/x/PyMS`. There are many algorithms for peak alignment, and PyMS uses a "Dynamic Programming" approach and is unique in this aspect.

The PyMS user guide which you can download from `code.google.com/pyms-docs` has information and a good reference on the dynamic programming algorithm. If you are interested in delving into specifics this is a good resource.

Today we will just aim to quickly align the experiments we have been working on.

1.7.1 The peak alignment script

The script `proc-align.py` aligns the peak lists using the PyMS dynamic programming algorithm. The peak lists have been saved into a PyMS object called an "Experiment". Our previous script `proc-expr.py` cropped the experiments so that the miminum and maximum masses are the same for each. Also the retention time range

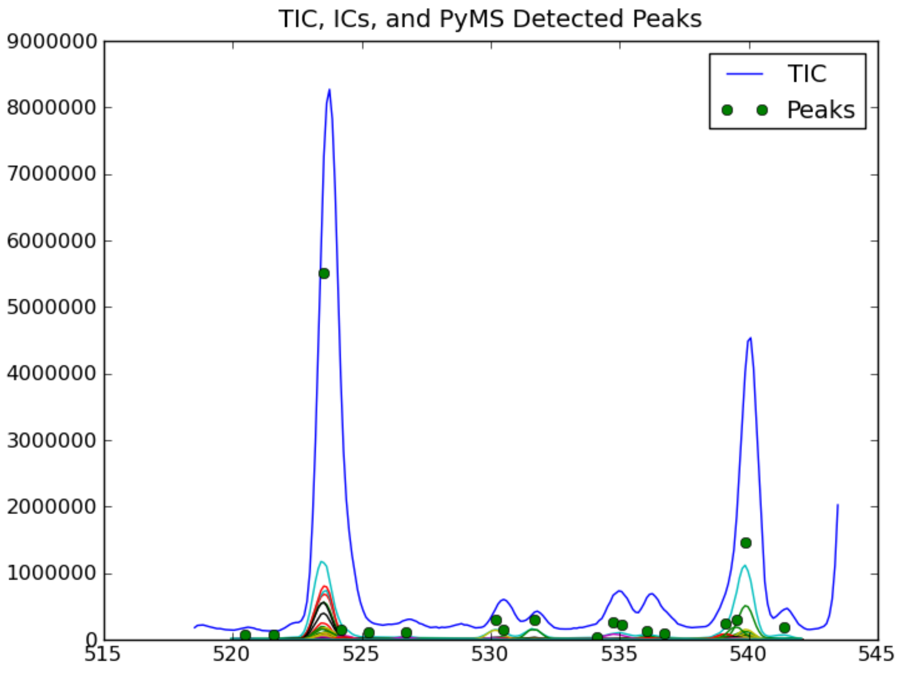

Figure 1.1: Graphics window displayed by the script proc-display-all.py

is the same for each experiment. The dynamic programming algorithm uses two parameters; the Gap Penalty which penalises the insertion of new gaps in the alignment matrix, and the retention time tolerance parameter which determines how important retention times should be for the alignment.

1.8 A script from scratch

In this section we will attempt to use PyMS to find the peaks in an area of a chromatogram with some coelution of peaks. The GC-MS data used here is from a mix of known metabolites, and the chromatogram has been interrogated using several approaches over a long period of time, so the peak list is known.

The area we will look at has a number of sugar peaks, one alkane, and a number of amino acids. I know the "true" peak list for this area and I would like you to try and find it using PyMS. We will start from scratch, writing the script as we go along.

1.8.1 Importing functions and files

Open a brand new file in gedit, and save it as proc.py. By saving it with this suffix at the very beginning gedit will be able to highlight Python syntax as we type.

First we need to tell the Python interpreter where the PyMS libraries can be found. These are of course in the `/x/PyMS/` directory. Using earlier scripts as a guide, set this value, and import the following functions which we will need to perform peak finding/deconvolution:

- `ANDI_reader`
- `build_intensity_matrix_i`
- `savitzky_golay`
- `tophat`
- `BillerBiemann, rel_threshold, num_ions_threshold`
- `Display`

1.8.2 Open and store the file

The file we need to look at is in the directory `data/` which is located in the current directory `/x/PyMS/`, and is called `GC01_0812_066.CDF`. We need to read this into a PyMS `data` object. We are only interested in the small area between scans 4101 and 4350, so use the command `data.trim(4101, 4350)` to discard the other points. Read what's left into an `intensity_matrix` object.

1.8.3 Filtering

Next we need to filter by Ion Chromatogram, and this can be done in the same way as in previous scripts.

1.8.4 Peak Finding

Now peak finding can begin, and we can try to use some sensible values for the parameters by taking a look at the TIC for this experiment. You will find the TIC in the `output/` directory where we previously looked at some TICs. Choose a sensible level for the threshold, and you can set some of the others to what we consider default values (`n = 3, scans = 2, r = 2`).

1.8.5 Results

Use a print statement to check how many peaks the BillerBiemann algorithm finds and compare this to what you might expect from looking at the TIC. Another more useful approach is to use the script `proc-plot-exercise.py` in the `workshop/` directory to plot all of the Ion Chromatograms rather than the TIC of this experiment. Now estimate how many peaks you think that PyMS should find. Modify your script if you deem it necessary.