gcmstools Documentation

Release 0.1.0

Ryan Nelson

January 11, 2015

gcmstools is a Python package that reads some GCMS file formats and does some simple types of fitting. The source code for this project can be found on GitHub. If you are reading this in PDF format, there is online documentation as well.

CHAPTER

ONE

GETTING STARTED

This user guide is broken into a few sections. First of all, there is information about getting a Python installation up and running. This is followed by a section on the basic usage of *gcmstools* to manipulate and plot GCMS data. There is also a section about fitting the GCMS datasets. The final section covers generating calibration curves and automating data extraction with this calibration information. You can skip to that section if all you want to do is automate some data extractions. (It is not necessary to understand the data manipulation/plotting as that is automated in the final section.)

1.1 Running Code Samples

Running the code samples in this documentation requires a rudimentary knowledge of using command-line terminal (command prompt on Windows). The terminal can seem very "texty" and confusing at first; however, with a little practice it gets to be fairly intuitive and *efficient*. There are many tutorials online. See The Command Line Crash Course, for example.

A few basics are covered there, though, for reference. When you start a terminal, you will be presented with a little window where you can type commands. In this documentation, the terminal command prompt will be denoted as `home>$` in this document. Where *home* is simply the current folder where the commands will be executed, and *>$* is just a separator. These things do not need to be typed when entering commands.

A couple of useful command line options are "change directory" (`cd`) and "present working directory" (`pwd`). The most important thing you'll want to be able to do is move directories (i.e. folders). When you open the terminal, you will usually start in your *home* directory. This will probably be "/Users/username" on Mac, "/home/username" on Linux, or "C:Usersusername" on windows. To move to a different directory, use the `cd` command: to find out the location of the current folder use `pwd`. Here's an example.

```
home$> pwd
/home/username/
home$> cd folder1
folder1$> pwd
/home/username/folder1
```

The second command here moved active directory to the folder "folder1". There are a few special directory shortcuts:

- `~` : This refers to the home directory.
- `..` : (Double period) This refers to the base directory of the current directory.
- `.` : (Single period) This refers to the current directory.
- `\` or `/` : Separators to combine directory names. The first works on Linux/Mac, the second is required on Windows.

Here's these shortcuts in action.

```
home>$ cd folder1/folder2
folder2>$ pwd
/home/username/folder1/folder2
folder2>$ cd .
folder2>$ pwd
/home/username/folder1/folder2
folder2>$ cd ../..
home>$ pwd
/home/username
home>$ cd folder1/folder2
folder2>$ cd ~
home>$ pwd
/home/username
home>$ cd folder1/folder2
folder2>$ cd ~/folder3
folder3>$ pwd
/home/username/folder3
```

The other important command is going to be "list" (`ls` or `dir` on Windows). This lists the contents of the current directory.

```
folder3>$ ls
file1 file2 folder4
```

CHAPTER

TWO

INSTALLATION

Gcmstools requires Python and a number of third-party packages. Below is a complete list of packages and minium versions:

- Python >=2.7 (3.x versions not yet supported)
- Pip >=6.0.6 (might be part of new Python releases)
- Setuptools >=11.3.1 (might be part of newer Python releases)
- Numpy >=1.9.1
- Matplotlib >= 1.4.2
- netCDF4 >=1.0.4
- PyTables >=3.1.1
- Scipy >=0.14.0
- Sphinx >=1.2.2 (Optional for documentation.)

Although not required, IPython (v 2.3.1 tested) provides a very useful advanced interactive Python interpreter, and examples in this documentation assume that you are using this environment.

2.1 Python

Python and the necessary packages can easily be installed using the all-in-one Anaconda Python distribution. It combines a large number of Python packages for scientific data analysis and a program (`conda`) for managing package updates (in addition to many other advanced features). The Anaconda developers (Continuum Analytics) have a lot of useful documentation for installing Anaconda and using conda. There are other ways to install Python and it's packages, but for this documentation, it will be assumed that Anaconda is being used.

Note: On Mac/Linux systems, Python is already part of the operating systems. Do not try to install these packages into the builtin Python distribution unless you really know what you are doing. You might overwrite and important file, which can cause problems for your system. Confusion between the system and Anaconda Python installation is a common source of problems for beginners, so make sure that your Anaconda Python is "activated" before running the commands in this document. (See the Anaconda documentation for more information on the activation process.)

Note: On Windows, Anaconda may not install netCDF4. In this case, you can get a prebuilt installer from Christoph Gohlke: be sure to get the Python 2.7 ("cp27") 64-bit ("amd64") build for the most recent version.

Learning the usage of all of these Python packages is far beyond the scope of this document. However, excellent documentation for most of the packages as well as full tutorials are easily discovered.

2.2 gcmstools

To install *gcmstools* from the main repository, there are two options: 1) download the source file and install the package or 2) install using `git` (recommended).

Option 1

Download a zip file of the current state of the repository. (Look for the button shown below (Figure 2.1)at the main repository.) Unzip this package wherever you'd like.

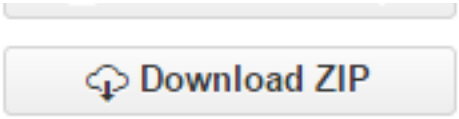

Figure 2.1: The zipfile download button.

From the command line, navigate the newly extracted folder and use `pip` to install the package. (In this case, *path-to-gcmstools-folder* is the location of the newly unzipped *gcmstools* folder.)

```
home>$ cd path-to-gcmstools-folder
gcmstools>$ pip install .
```

Option 2 (recommended)

First, install the version-control software Git. Now, download and install *gcmstools* with one command.

```
home>$ pip install git+https://github.com/rnelsonchem/gcmstools.git
```

The advantage here is that the same command will update your *gcmstools* installation with any any changes that have been made to the main repository.

Uninstall

Uninstallation of *gcmstools* is identical regardless of the installation method used above.

```
home>$ pip uninstall gcmstools
```

CHAPTER

THREE

BASICS OF WORKING WITH GCMS DATA FILES

3.1 Export the data

First of all, be sure to export your GCMS data in a common data format, such as AIA, ANDI, or CDF. It turns out that all of these formats are related in that they are all based off of Network Common Data Format (netCDF), so they may have the file extension "AIA" or "CDF". This file type may not be the default for your instrument, so consult the documentation for your GCMS software to determine how to export your data in these formats.

3.2 Set up the processing environment

In order to process these files, run IPython from a terminal (command prompt in Windows) in the folder containing the "gcms.py" file (which is the base folder for this repository). There are (at least) two ways to do this that involve the command `cd` (change directory) run from either the terminal or an IPython session. For example (`home>$` is the command prompt located in the folder "home", `In:` is the IPython prompt):

```
home>$ ipython
In: %cd "path-to-gcms-folder"
Out: path-to-gcms-folder
In:
```

or:

```
home>$ cd path-to-gcms-folder
gcms>$ ipython
In:
```

The "*path-to-gcms-folder*" is a valid path to the folder with "gcms.py". It can take a little practice, but this gets easier very quickly. I'll assume you use the second form of this as it makes using the IPython notebook much easier later.

3.3 Read AIA data files

First of all, you will need to `import` the "gcms.py" file to make the code accessible to the IPython environment. This file contains a class called `AIAFile` that reads and processes the GCMS files. AIAFile takes one argument, which is a string with the file name. This string must have the path (i.e. folder) information if the file is not in the same directory as "gcms.py". Sample data files are contained in a folder called "data".

```
In: import gcms
In: data = gcms.AIAFile('data/datasample1.CDF')
```

The variable `data` now contains our processed GCMS data set. You can see its contents using tab completion in IPython (`<tab>` refers to the tab key).

```
In: data.<tab>
data.filename data.intensity data.nnls data.ref_build data.times
data.integrate data.masses data.tic
```

All of these attributes are either data that describe or functions that modify (methods) our dataset. You can inspect these attributes very easily in IPython by just typing the name at the prompt.

```
In: data.times
Out:
array([0.08786667, ..., 49.8351])
In: data.tic
Out:
array([158521., ..., 0.])
```

This is a short description of these initial attributes:

- *filename*: This is the name of the file that you imported.
- *times*: A Numpy array of the times that each MS was collected.
- *tic*: A Numpy array of the total ion chromatogram intensities.
- *masses*: A Numpy array the masses that cover the data collected by the MS.
- *intensity*: This is the 2D Numpy array of raw MS intensity data. The columns correspond to the masses in the `masses` array and the rows correspond to the times in the `times` array.

The remaining attributes *ref_build*, *nnls*, and *integrate* are functions that deal with the non-negative fitting routine and are covered in later sections.

3.4 Simple plotting

We can easily plot these data using the plotting package Matplotlib. As an example, let's try plotting the total ion chromatogram. In this case, `data.times` will be our "x-axis" data, and `data.tic` will be our "y-axis" data.

```
In: import matplotlib.pyplot as plt
In: plt.plot(data.times, data.tic)
Out:
[<matplotlib.lines.Line2D at 0x7f34>]
In: plt.show()
```

This should produce a pop-up window with an interactive plot, Figure 3.1. (This should process should be fairly quick. However, sometimes the plot initially appears behind the other windows, which makes it seem like things are stuck. Be sure to scroll through your windows to find it.)

One drawback here is that you have to type these commands every time you want to see this plot. There is another alternative, though. You can also put all of these commands into a text file and run it with Python directly. Copy the following code into a plain text file called "tic_plot.py".

NOTE: it is very important that you are using a plain text file and not a word processing (MS Word) document. On Mac/Linux, the ".py" suffix is not required; however, in Windows, this suffix can be important. Unfortunately, Windows hides file extentions by default, so you may have to search the web to determine how to enable display of file extensions. Otherwise, you might end up with a file called "tic_plot.py.txt", which can work, but will most likely cause confusion and annoyance. Anaconda ships with Spyder, a Python development editor, which will take care of all of this for you, so you might want to familiarize yourself with that program.

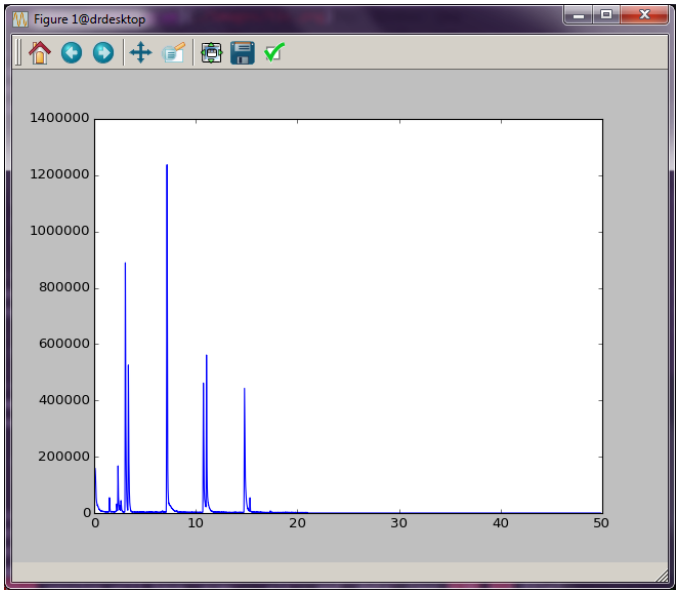

Figure 3.1: Total ion chromatogram.

```
import matplotlib.pyplot as plt
import gcms

data = gcms.AIAFile('data/datasample1.CDF')
plt.plot(data.times, data.tic)
plt.show()
```

It is common practice to do all imports at the top of a Python program. That way it is clear exactly what code is being brought into play. Run this new file using the `python` command from the terminal.

```
gcms>$ python tic_plot.py
```

The window with your plot will now appear. (You will not be able to work in the terminal until you close this window.) Alternatively, you can run this program directly from IPython.

```
gcms>$ ipython
In: %run tic_plot.py
```

This also pops open a new window containing the interactive plot. It has the advantage, however, that once the window is closed, you are dropped back into an IPython session that "remembers" all of the variables and imports that you created in your program file. In our example above, once the plot window is closed, your IPython session will have `gcms`, `plt`, and `data` (our GCMS AIA file) available. This is very useful if you want to continue to work interactively with your data, and it is a great way to remove a bunch of repetitive typing.

3.5 Working with multiple data sets

In the example above, we opened our dataset into a variable called `data` in order to be able to plot the TIC. If you want to manipulate more than one data set, the procedure is exactly the same, except that you will need to use different variable names for your other data sets.

```
In: data2 = gcms.AIAFile('data/datasample2.CDF')
```

These two data sets can be plot together on the same figure by doing the following:

```
In: plt.plot(data.times, data.tic)
Out:
[<matplotlib.lines.Line2D at 0x7f34>]
In: plt.plot(data2.times, data2.tic)
Out:
[<matplotlib.lines.Line2D at 0x02e3>]
In: plt.show()
```

The window shown in Figure 3.2 should appear on the screen. (There is a blue and green line here that are a little hard to see in this picture. Zoom in on the plot to see the differences.)

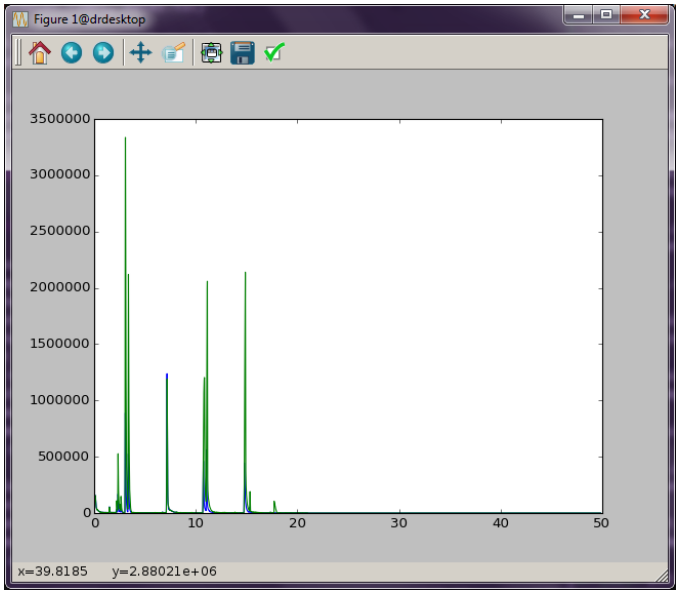

Figure 3.2: Two tic plotted together

CHAPTER

FOUR

NON-NEGATIVE LEAST SQUARES FITTING

4.1 Collecting Reference MS Files

A series of reference spectra are required if you want to do non-negative least squares (NNLS) fitting of your data. There are two example files in this repository to use as reference: "ref_spec.txt" and "ref_spec2.MSL". The MSL file is a type of text file library that can be exported by programs like the AMDIS or the NIST Mass Spectral Database. The .txt file was hand generated; more information on this file format is provided in the next paragraph. The format of both of these files is very important; however, MSL files are typically autogenerated by external software and may not need manual modification. A common feature of both formats, though, is that comment lines can be included by starting a line with #. This can be useful if you want to add some notes or to remove reference spectra without deleting them entirely.

Hand generated reference files must be text files that end with the prefix ".txt". Each reference compound must have at the minimum two labels: "NAME" and "NUM PEAK". "NAME" will be the reference name (probably want this to be concise), and "NUM PEAKS" is followed by (at least) two space-separated columns of MS data. The first column are m/z values, and the second column are the intensity values. Intensities are normalized on import; it is not necessary to do this by hand. Other labels can also be included if you would like to incorporate extra metadata about the reference compound. Each reference compound *must* be separated by a blank line. Below is a small sample of one of these files:

```
NAME:octane
FROM:www.massbank.jp
ID_NUM: JP004695
NUM PEAKS:
  42 14.07 141
  43 99.99 999
  44 2.54 25
  45 4.03 40
  53 1.58 16
  55 19.83 198
  .
  .
  .
```

The online MS repository massBank is a useful place to find these mass and intensity values. The data from that site is already formated correctly for this file type.

4.2 Loading Reference Spectra

In order to import this reference file, we will use the AIAFile object's `ref_build` function. Below is a repeat of the steps necessary in IPython to set up our environment. (These are unchanged from the previous sections but are repeated for clarity.)

```
In: import matplotlib.pyplot as plt
In: import gcms
In: data = gcms.AIAFile('data/datasample1.CDF')
```

At this point, we are ready to read in our reference file using the function `AIAFile.ref_build`.

CHAPTER

FIVE

AUTOMATED CALIBRATION AND INTEGRATION

5.1 Calibration Data

If you have calibration data for a particular reference compound, you must create a csv file and folder that have the same base name as the reference MS file from above. Again, an examples are provided in this repository called refcpd.csv and the folder refcpd. All of your calibration AIA files for this compound need to be stored in the newly created folder. In order for these new data files to be processed, the refcpd.csv file must be appropriately modified.

The csv file is a simple comma-separated text file, but again the structure is important. The first row in this file is critical. At the end, there are two values that define the starting and stopping time points for integration. Change these values based on the time range that you've determined from the TIC of a calibration run. The rest of the rows are data file information. The first column is the name of a calibration data file, and the second column needs to be the concentration of the reference compound associated with that run. You don't have to add all of the calibration files here, but if they are not in this list, they won't be processed. Alternatively, any line that starts with a '#' is a comment, and will be ignored. In this way, you can comment out samples, and add some notes as to why that sample was not used or whatever.

5.2 Run Calibrations

Once you've updated the calibration information from above. You can run the program 'calibration.py'. This runs through all of the reference spectra defined in the 'reference_files.txt' file. If a '.csv' file exists for a particular reference file, then a calibration will be performed.

All of the calibration data files listed in the csv file will be processed and a calibration curve generated. For each calibration sample, a plot of the reference-extracted data will be generated in the calibration folder (refcpd_fits.png). In addition, a calibration curve plot is also generated (refcpd_cal_curve.png'), which plots the integrated intensities and calibrated intensities vs the concentrations. In addition, the calibration information is printed on the graph for quick visual inspection. There is no need to write down this calibration information.

This program has some important command line arguments that will change the programs defaults. The first argument, '–nobkg', is a simple flag for background fitting. By default, the fitting routine will select a MS slice from the data set and use that as a background in the non-negative least squares fitting. This procedure can change the integrated values. If you use this flag, then a background MS will not be used in the fitting. Using a background slice in the fitting may or may not give good results. It might be a good idea to look at your data with and without the background subtraction to see which is better.

The second command line argument is '–bkg_time'. By default, the fitting program uses the first MS slice as a background for fitting. However, if there is another time that looks like it might make a better background for subtraction, then you can put that number here.

Here's a couple of example usages of this script:

```
# This will run the calibration program with all defaults
$ python calibration.py
# This shuts off the background subtraction
$ python calibration.py --nobkg
# This sets an alternate time for the background subtraction
# In this case, the time is set to 0.12 minutes
$ python calibration.py --bkg_time 0.12
```

Another file is also generated during this process: cal.h5. This is a HDF5 file that contains all of the calibration information for each standard. Do not delete this file; it is essential for the next step. This is a very simple file, and there are many tools for looking at the internals of an HDF5 file. For example, ViTables is recommended. The background information, such as whether a background was used and the time point to use as a background spectrum, are stored as user attributes of the calibration table.

CHAPTER

SIX

PROCESS SAMPLE DATA

Put all of your data files in a folder that must be called ‘data’. Once you’ve done this, run the program ‘data.py’ to process every AIA data file in that folder using the calibration information that was determined from the steps above.

This program opens the AIA file for the sample and performs non-negative least squares analysis of the full data set using the reference spectra that are listed in the ‘reference_files.txt’ file. Using the calibration information that was determined above, it finds the concentrations of those components in the sample data. For every reference compound that has associated calibration information, a plot is generated that overlays the TIC (gray) and extracted reference fit (blue). The title of the plot provides the calibrated concentration information. Visual inspection of these files is recommended.

This file also accepts the same command line arguments as ‘calibration.py’ from the section above. You will be warned if you try to analyze your data with different background information than the calibration samples. This may not impact your data much, but it is good to know if you are doing something different.

This file also generates another HDF5 file called ‘data.h5’, which contains the integration and concentration information for every component. This information is identical to what is printed on the extraction plots above. However, this tabular form of the data is a bit more convenient for comparing many data sets. See the Calibration section for a recommended HDF5 file viewer.