The PSRCHIVE Python Interface

Paul Demorest, 2018/03/28

IPTA 2018 Student Workshop, Socorro, NM

- Background information: What is it; how does it work; why would you want to use it?
- How to use it: Installation; basic usage; overview of the PSRCHIVE class structure
- Some simple examples
- Activities: See handout for suggestions; also anything else you may be interested in working on.

Background, motivations

- The PSRCHIVE Python interface lets you access a subset of the PSRCHIVE classes (data structures, subroutines) directly from Python.
  - Lower-level than the command line utils (eg. pam, pat, psredit, etc).
  - See http://psrchive.sourceforge.net/manuals/python
- Why is this useful?
  - Direct access to data values for exploration, debugging, etc.
  - Prototyping or implementation of new analysis routines (often easier in Python than C++).
  - More complex scripting than is possible with psrsh.
  - More flexible and/or prettier plotting than is possible with pav/psrplot.
- When this is not so useful – reproducing complex PSRCHIVE applications (pac, pat).

How does it work

- Built using the “Simplified Wrapper and Interface Generator” aka SWIG.
  - SWIG examines the PSRCHIVE C++ header code and automatically generates ~40,000 lines of “wrapper” code to allow calling the C++ routines from Python.
  - Can in principle generate bindings for languages besides Python.
  - http://swig.org

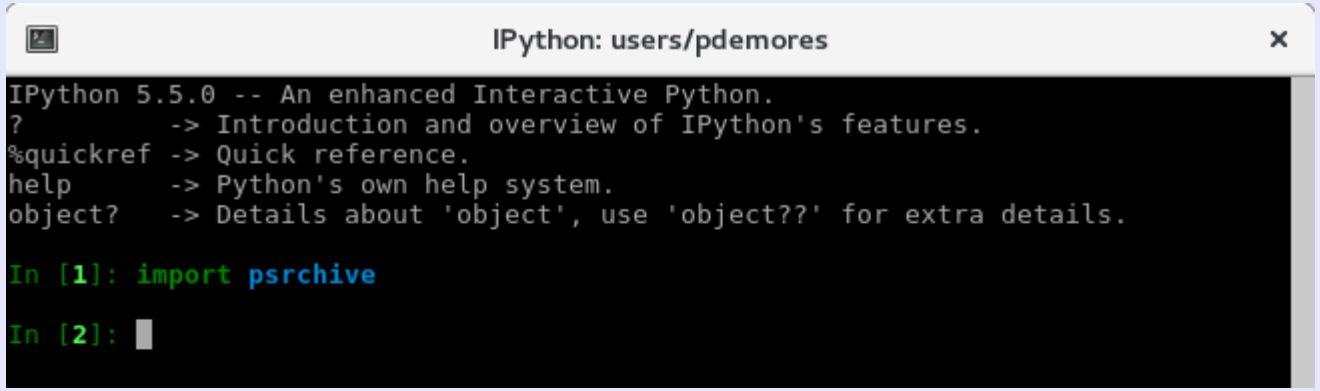
How does it work

- Built using the “Simplified Wrapper and Interface Generator” aka SWIG.
  - SWIG examines the PSRCHIVE C++ header code and automatically generates ~40,000 lines of “wrapper” code to allow calling the C++ routines from Python.
  - Can in principle generate bindings for languages besides Python.
  - http://swig.org
- “SWIG is lame. Why not use [*flavor of the month C/python interface*] instead?”
  - The SWIG interface has existed/worked for a long time; so, some inertia.
  - Even today, I have not found very many good options for *automatic* wrapper generation. But please let me know if you have suggestions!

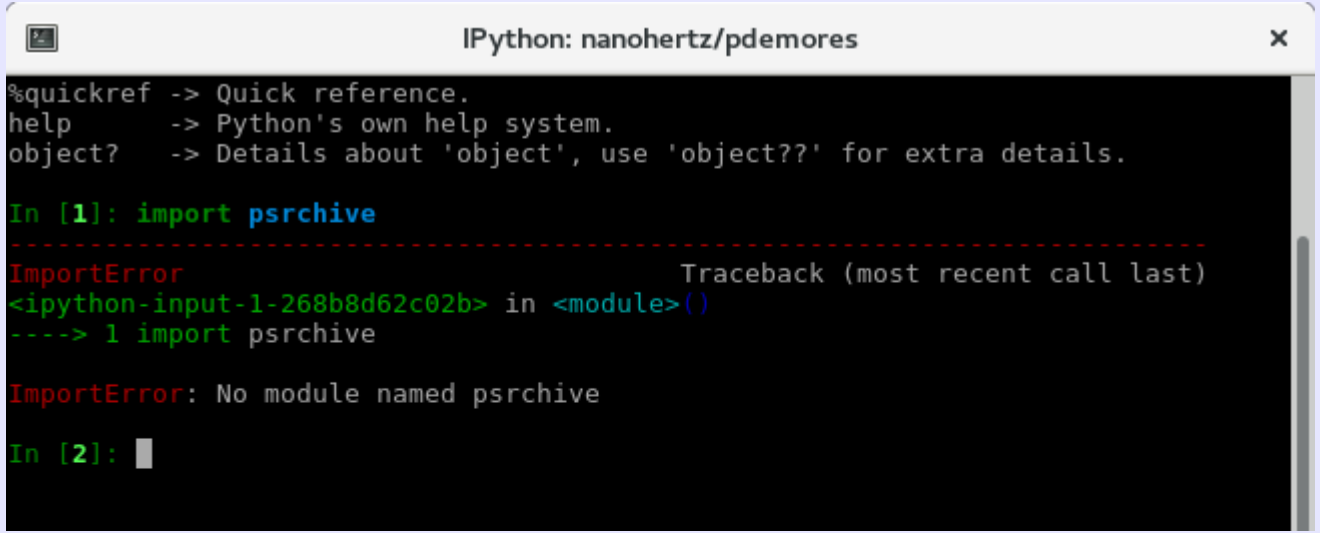
Installation

- Nate has already bundled this into the Docker image (`source activate python2` first). But, in case you ever need to do your own install:
- Python wrapper is distributed with PSRCHIVE; no additional download.
- Basic requirements (beyond those of standard PSRCHIVE):
  - Python, with development headers (“python-dev” or similar pacakge).
  - SWIG; note, some reports of problems with v3.x
  - NumPy
- Very useful but not required: SciPy, matplotlib, ipython/jupyter
- PSRCHIVE builds via “`configure; make; make install`” process.
  - Make sure you configure with `--enable-shared`
  - If all above requirements met, wrappers will be generated!

Checking that it's working:



Good!



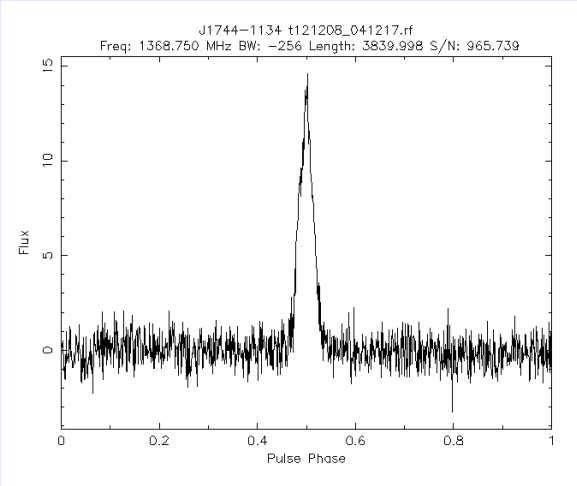
Bad!

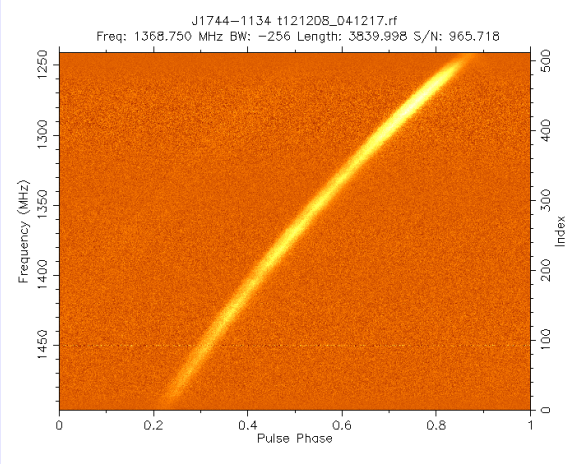
A super-simple example:

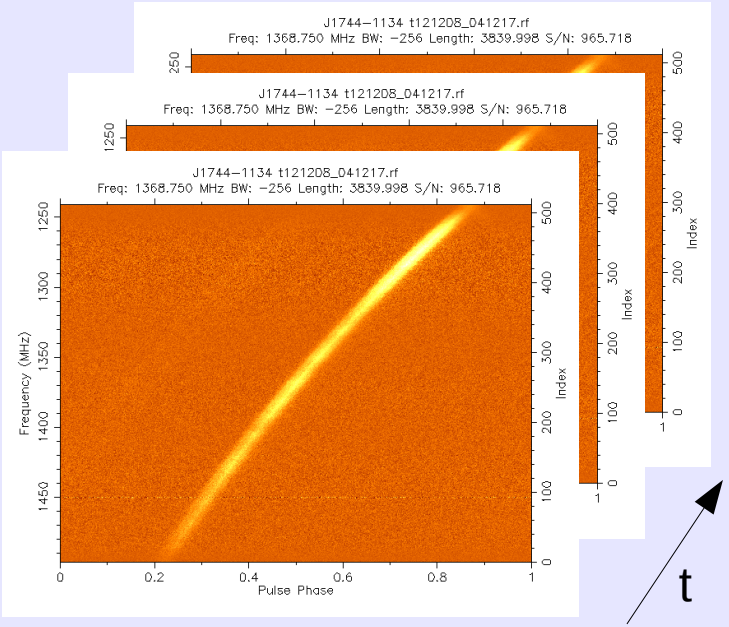
```
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import psrchive

In [2]: arch = psrchive.Archive_load('t121208_041217.rf')

In [3]: arch.get_source()
Out[3]: 'J1744-1134'

In [4]: arch.get_nsubint()
Out[4]: 64

In [5]: arch.get_nchan()
Out[5]: 512

In [6]:
```

The three fundamental PSRCHIVE data classes

- `Profile` is a single pulse profile – data as a function of pulse phase only.
- `Integration` is a set of pulse profiles recorded simultaneously – usually profiles as a function of frequency channel and/or polarization.
- `Archive` is a set of `Integration` as a function of time. Represents a single data file.



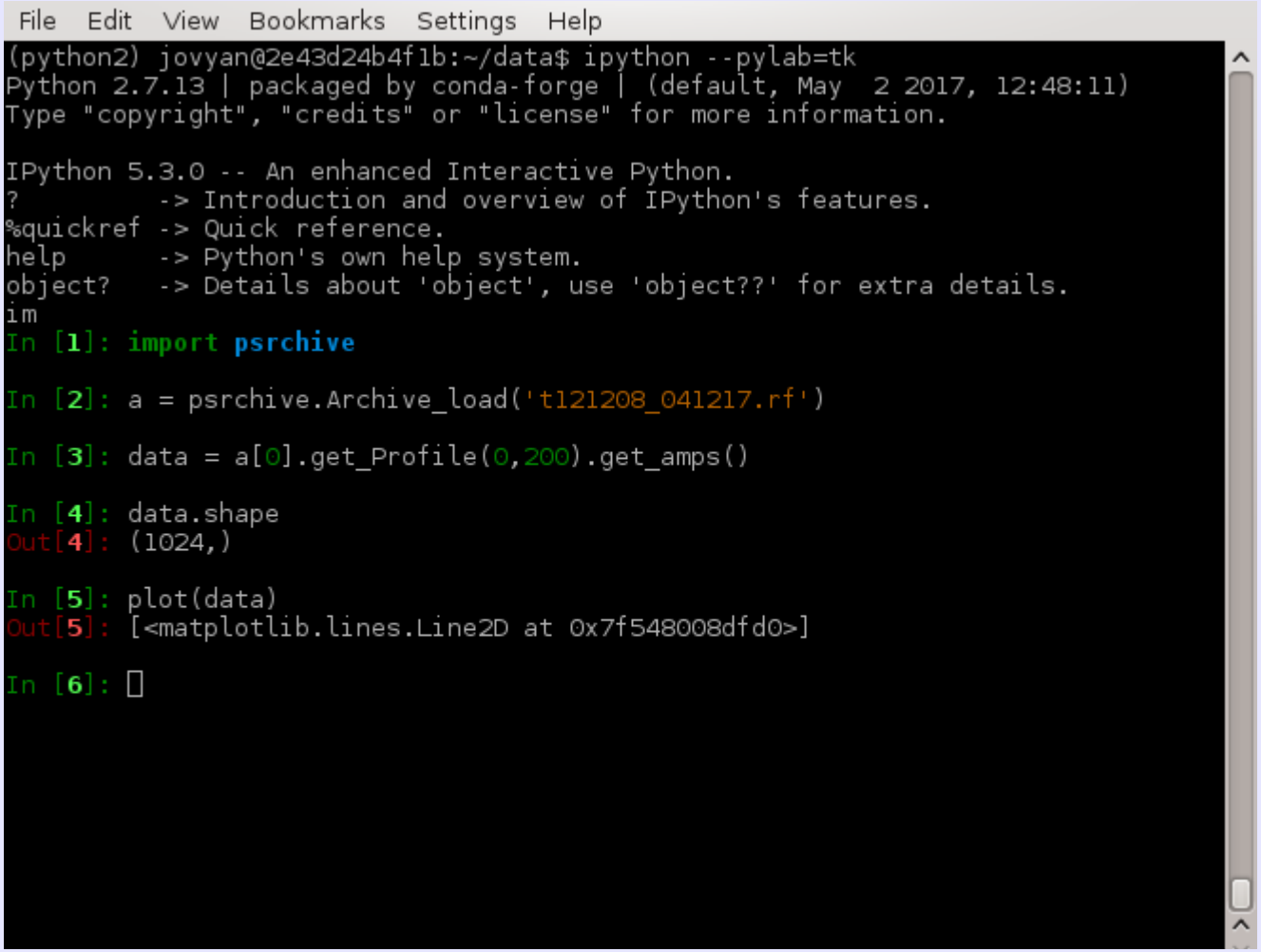




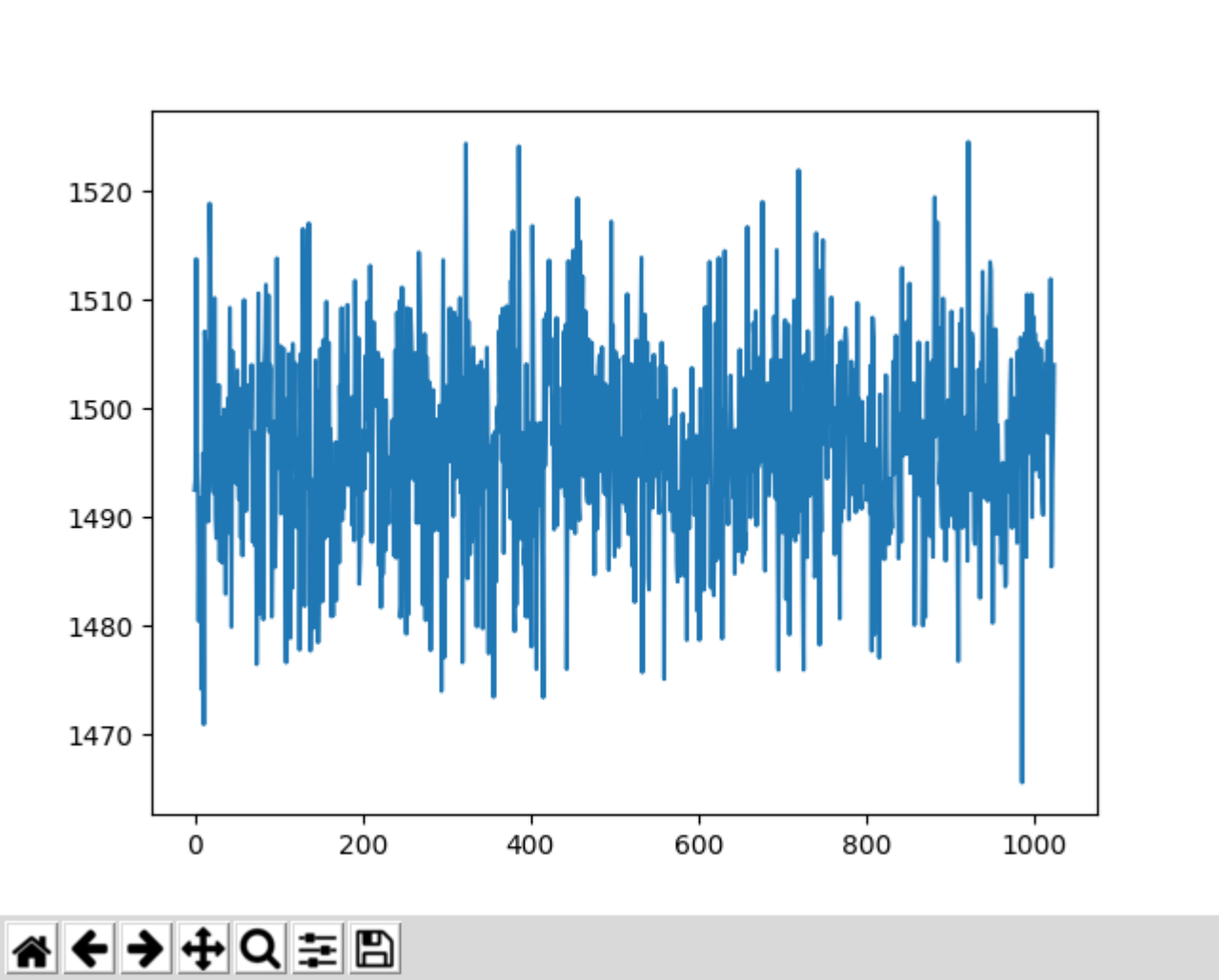
Accessing data in Python using these classes

- `Archive`:
  - use `archive.get_Integration(isub)`
  - or `archive[isub]` to retrieve an Integration
- `Integration`:
  - use `integration.get_Profile(ipol,ichan)` to retrieve a single Profile
- `Profile`:
  - use `profile.get_amps()` to return data as a NumPy array
- Shortcut to get all data:
  - Use `archive.get_data()` to return entire (N_{sub}, N_{pol}, N_{chan}, N_{bin}) data cube as a NumPy array.

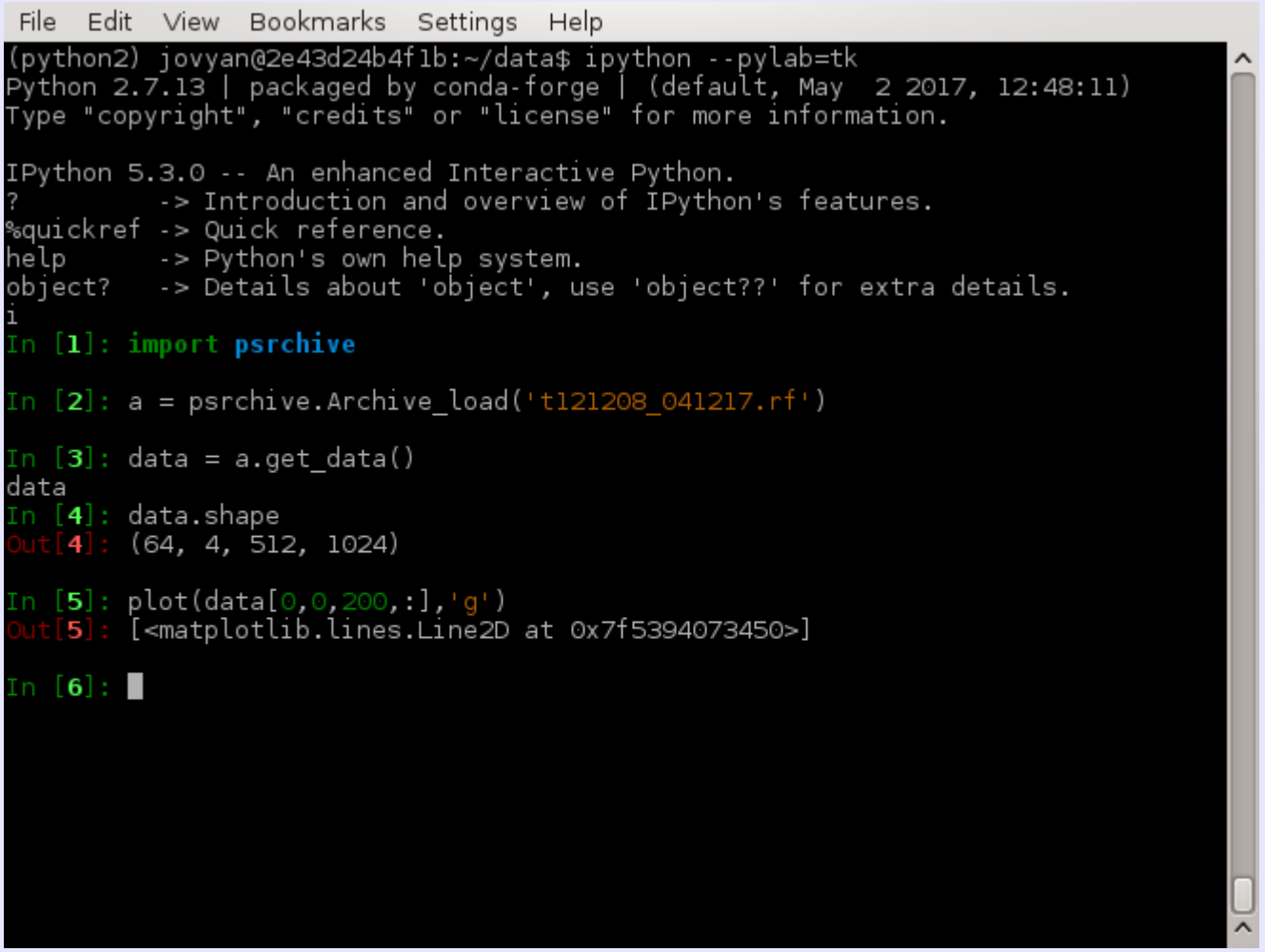
Plotting example:



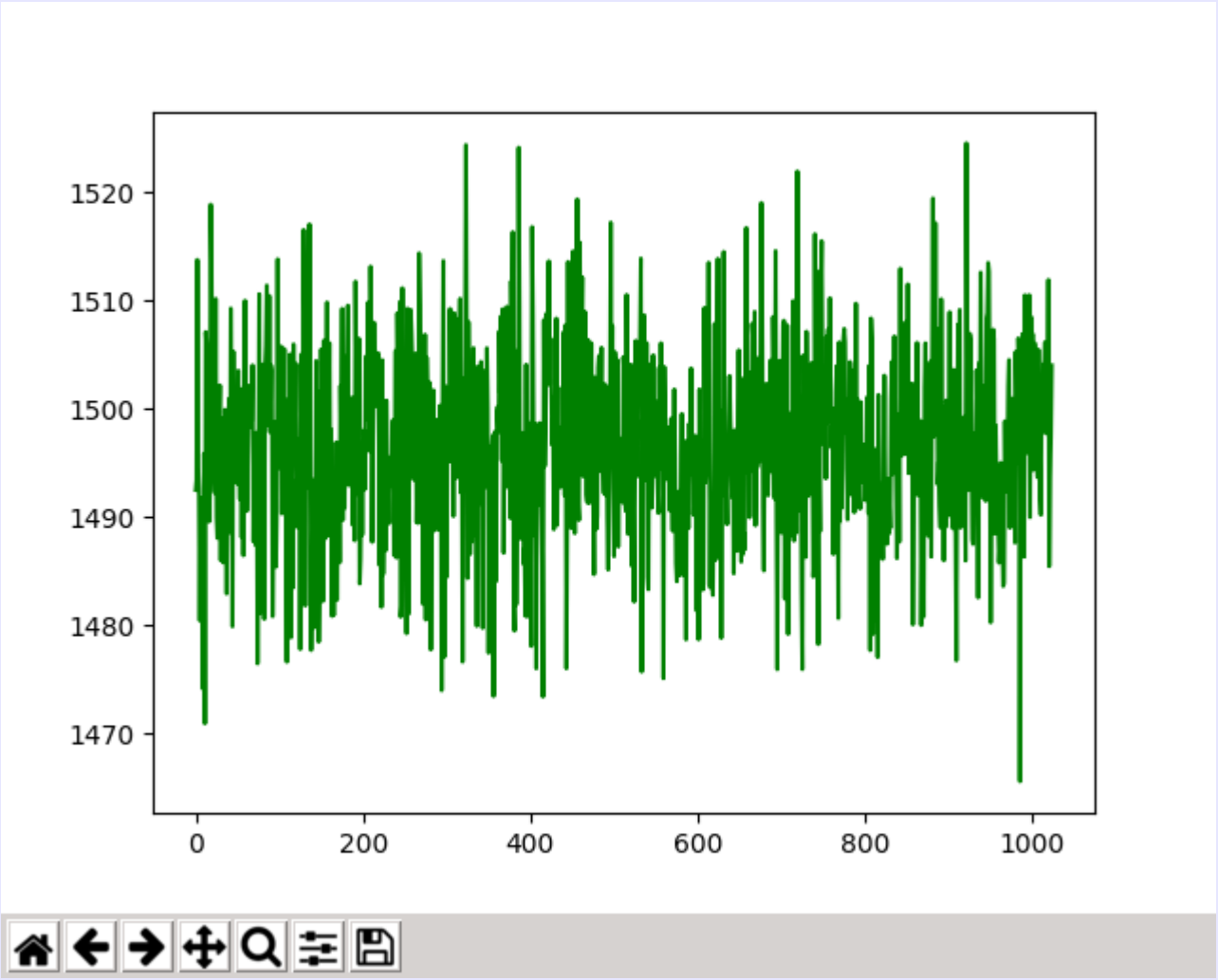
Plotting example:



Plotting example:



Plotting example:



Accessing data using these classes

- One data access subtlety / gotcha:
  - `profile.get_amps()` returns a *view* of the original data
  - `archive.get_data()` returns a *copy* of the original data
- This means that if you want to modify the data in a file you need to change the values in the results of `profile.get_amps()`
- Modified data files can be saved to disk using `archive.unload(“new_filename”)`

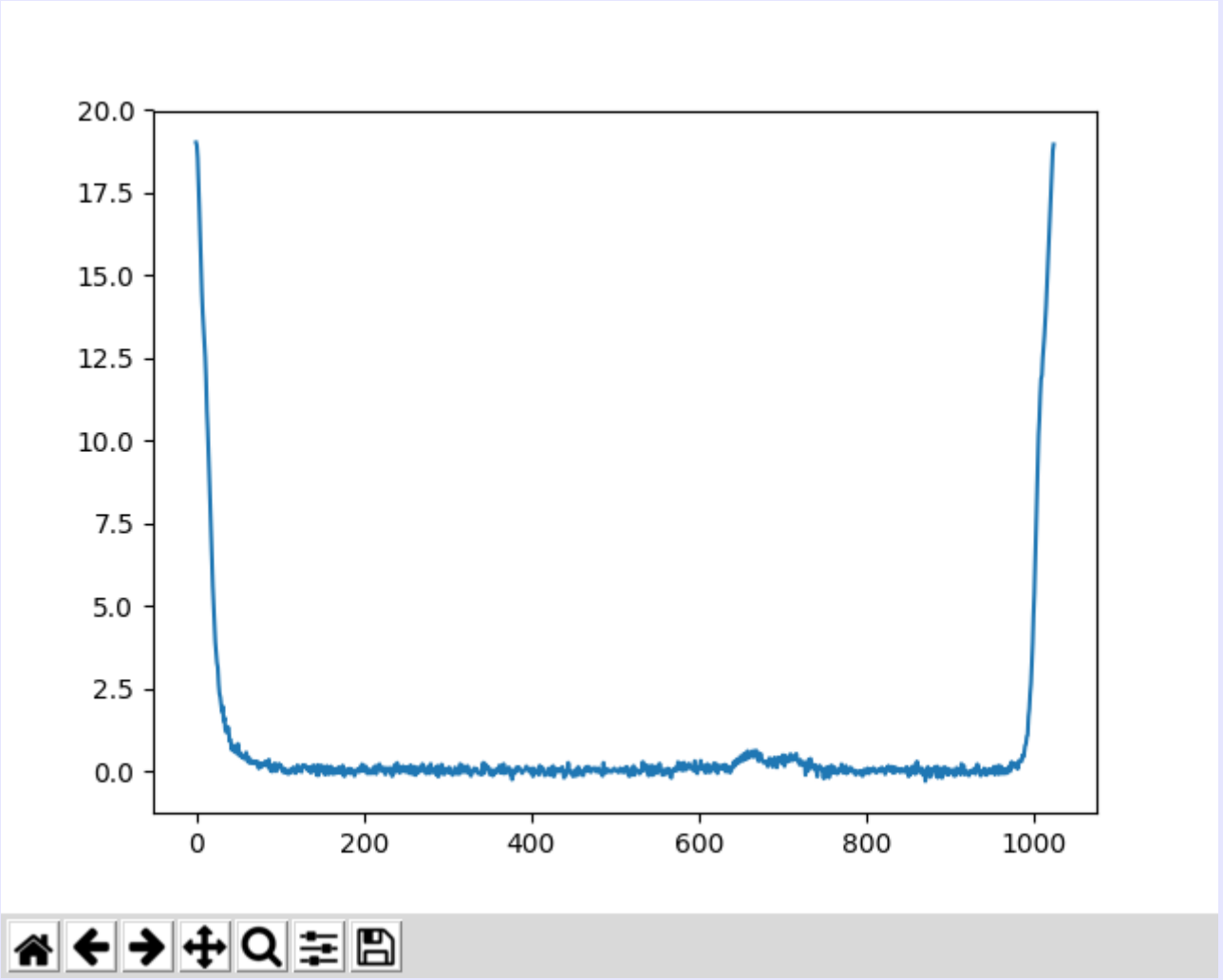
Data processing methods

- `Archive` has a large number of methods (functions) for performing common data processing steps.
  - Common examples: `dedisperse()`, `remove_baseline()`, `fscrunch()`, `tscrunch()`, `pscrunch()`, `convert_state()`, …
  - `archive.execute("[`*`psrsh code`*`…]")` will run any psrsh command on the archive.
- How to learn what else is available?
  - Browse the PSRCHIVE class documentation at http://psrchive.sourceforge.net/classes/psrchive
  - Tab completion in ipython is *very* useful!

Plotting example with pre-processing:

```
(python2) jovyan@2e43d24b4f1b:~/data$ ipython --pylab=tk
Python 2.7.13 | packaged by conda-forge | (default, May  2 2017, 12:48:11)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import psrchive

In [2]: a = psrchive.Archive_load('t121208_041217.rf')

In [3]: a.tscrunch()

In [4]: a.dedisperse()

In [5]: a.fscrunch()
a.
In [6]: a.pscrunch()

In [7]: a.remove_baseline()

In [8]: data = a[0].get_Profile(0,0).get_amps()

In [9]: plot(data)
Out[9]: [<matplotlib.lines.Line2D at 0x7f7598e7f0d0>]

In [10]:
```

Plotting example with pre-processing:



Looping over profiles, extracting data:

File Edit View Bookmarks Settings Help

```
In [1]: import psrchive

In [2]: a = psrchive.Archive_load('t121208_041217.rf')

In [3]: for isub in range(a.get_nsubint()):
   ...:     i = a[isub]
   ...:     for ichan in range(a.get_nchan()):
   ...:         print isub, ichan, i.get_Profile(0,ichan).get_amps()[0]
   ...:
0 0 747.418
0 1 763.687
0 2 772.33
0 3 791.966
0 4 819.075
0 5 846.44
0 6 863.161
0 7 893.33
0 8 935.528
0 9 961.15
0 10 1006.07
0 11 1053.43
0 12 1076.77
0 13 1108.51
0 14 1139.99
0 15 1181.05
0 16 1217.86
0 17 1239.19
0 18 1261.8
0 19 1274.39
0 20 1279.23
0 21 1296.06
0 22 1328.82
```

Other PSRCHIVE classes:

Most use cases only need `Archive`, `Integration`, and `Profile` classes.

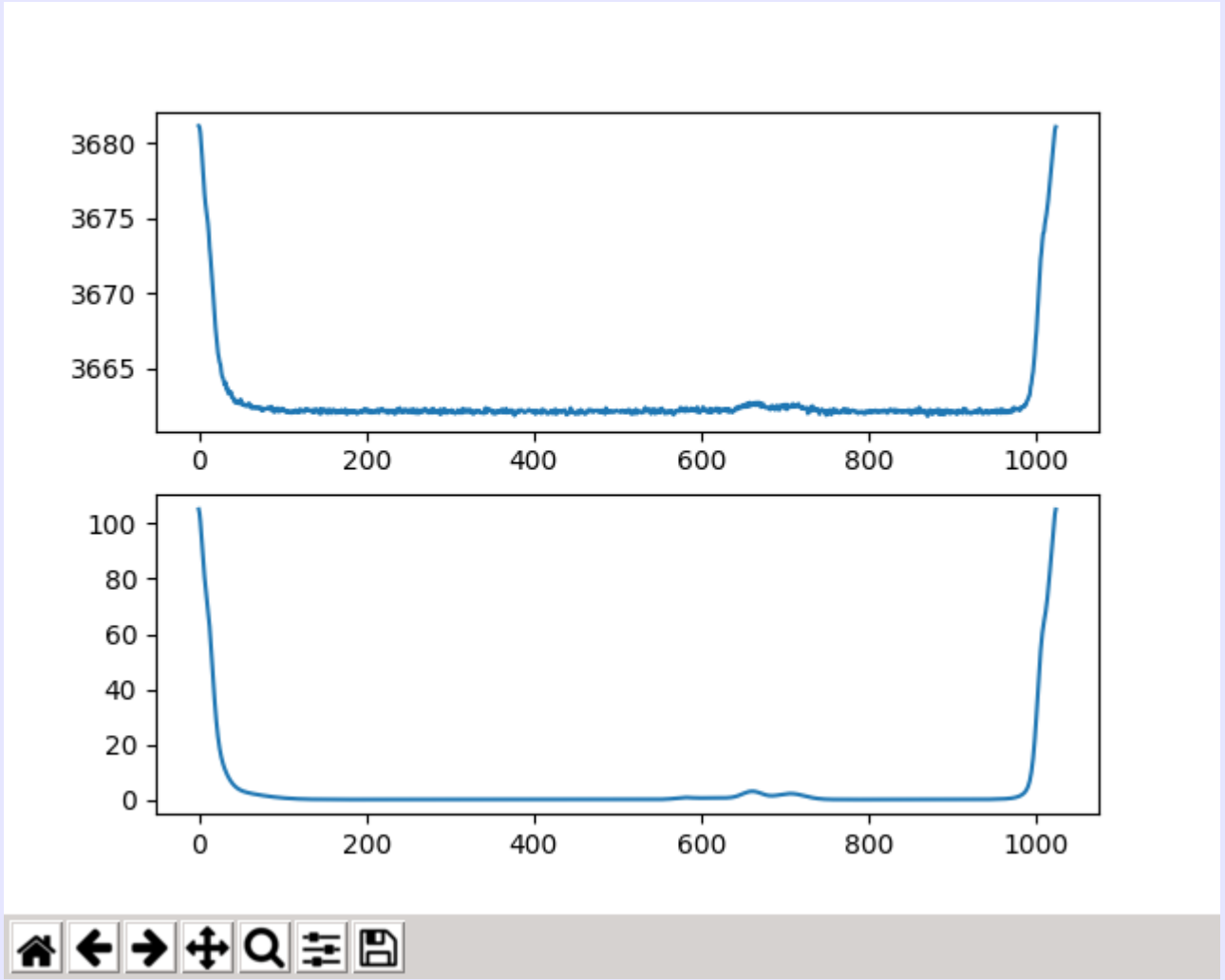
But some of the PSRCHIVE algorithm classes are also included in the Python interface.

For example, `ProfileShiftFit` for doing template-matching:

No very comprehesive list of these unfortunately. Browse the C++ class docs, let me know if you want something added.

```
File  Edit  View  Bookmarks  Settings  Help
IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import psrchive

In [2]: a = psrchive.Archive_load('t121208_041217.rf').total()

In [3]: s = psrchive.Archive_load('J1744-1134_20cm_ana_PDFB4.std')

In [4]: aprof = a[0].get_Profile(0,0)

In [5]: sprof = s[0].get_Profile(0,0)

In [6]: psf = psrchive.ProfileShiftFit()

In [7]: psf.set_standard(sprof)

In [8]: psf.set_Profile(aprof)

In [9]: psf.get_shift()
Out[9]: (-0.00014555419416436232, 4.3622893898029965e-10)

In [10]: subplot(211); plot(aprof.get_amps())
Out[10]: [<matplotlib.lines.Line2D at 0x7fbad00b1690>]

In [11]: subplot(212); plot(sprof.get_amps())
Out[11]: [<matplotlib.lines.Line2D at 0x7fbaca312f50>]

In [12]:
```

Advanced PSRCHIVE classes:



Data profile

Standard (aka template) profile

Summary

- PSRCHIVE has a SWIG-based interface to Python
- This allows you direct access to a large fraction of PSRCHIVE C++ classes via Python
- This is useful for directly exploring data; prototype or implement new algorithms; make custom plots; etc.
- See the handout for some simple (as well as less simple) exercises.