Accessing LYRA Data With Python

Daniel F. Ryan

November 14, 2014

Part I

Acessing LYRA Data With Python

LYRA data is openly available for download at http://proba2.oma.be/data/LYRA. The three principal data products are level 1, level 2 and level 3 FITS file. Level 1 contains uncalibrated and engineering data. Level 2 contains calibrated data at the observed time cadence. Level 3 contains calibrated 1-minute-averaged data derived from level 2. Here we will focus mainly focus on level 2.

Another important data product is the LYRA Time Annotation Files, containing information on data artifacts. This will be briefly mentioned at the end of this guide.

Before accessing LYRA data with Python, it is recommended that you install SunPy and all neccessary dependencies. To do this, see the SunPy installation guide at http://docs.sunpy.org/en/stable/guide/installation/index.html.

1 Using The SunPy LightCurve Object

In [1]:
```
%matplotlib inline
```

The simplest way to access and manipulate LYRA data is via the SunPy LightCurve object. For more on the SunPy LightCurve object, see http://docs.sunpy.org/en/stable/code_ref/lightcurve.html.

The SunPy LightCurve object is currently undergoing a revamp to improve usability and functionality. Any users who would rather access the data in a more manual way are encouraged to read the section, Maunally Accessing LYRA Data, below.

For now, however, we will explore the LYRALightCurve object. Once SunPy and neccessary dependencies are installed you can create a LYRALightCurve object in the following way:

In [2]:
```
import sunpy
```

In [3]:
```
import sunpy.lightcurve as lc
```

```
/Users/danielr/anaconda/lib/python2.7/site-
packages/glymur/lib/config.py:154: UserWarning: Neither the openjp2
nor the openjpeg library could be loaded.  Operating in severely
degraded mode.
```

```
    warnings.warn(msg, UserWarning)
```

In [4]:
```
lyra = lc.LYRALightCurve.create("2011-06-07")
```

```
/Users/danielr/git/sunpy/sunpy/lightcurve/lightcurve.py:253:
RuntimeWarning: Using existing file rather than downloading, use
overwrite=True to override.
  warnings.warn("Using existing file rather than downloading, use
overwrite=True to override.", RuntimeWarning)
```

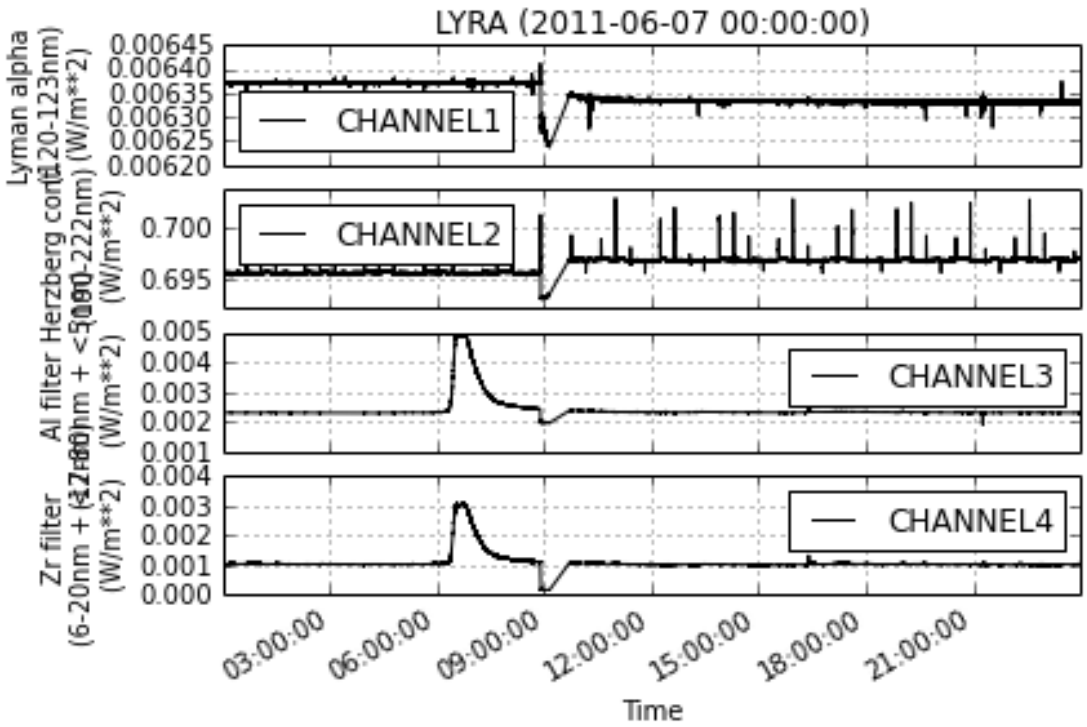
This creates a LYRALightCurve object, called lyra, which contains the data and header information of the level 2 FITS file for the given date. Currently, LYRALightCurve can only deal with level 2 data. However, this is being changed as part of the LightCurve revamp. For now, those wishing to access level 3 data should read the section, Manually Accessing LYRA Data, below.

1.1 Creating a Quick Look of the Data

To see a quick look version of the data, using the peek() attribute. To do this, you may need to import the matplotlib package.

In [5]:
```
fig = lyra.peek()
```

```
/Users/danielr/anaconda/lib/python2.7/site-
packages/matplotlib/figure.py:371: UserWarning: matplotlib is
currently using a non-GUI backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



More customised plots, e.g. for papers, can be generated using the matplotlib package. Matplotlib is a dependency for SunPy, but can be installed independently. For more information on how to use matplotlib, see http://matplotlib.org/.

1.2 Accessing Meta Data

The header information is stored in and meta attributes of the object. This can viewed by:

In [6]:
```
lyra.meta
```

Out [6]:
```
OrderedDict([('SIMPLE', True), ('BITPIX', 8), ('NAXIS', 0), ('EXTEND',
True), ('ORIGIN', 'ROB'), ('TELESCOP', 'PROBA2'), ('INSTRUME',
'LYRA'), ('OBJECT', 'EUV solar irrad'), ('OBS_MODE', 'standard'),
('DATE', '2012-10-12'), ('DATE-OBS', '2011-06-07T00:00:00.010000'),
('DATE-END', '2011-06-07T23:59:59.988995'), ('DATASRC', 'Redu'),
('LEVEL', '2'), ('ALGOR_V', 'EDG=2.1  BSDG=0.8'), ('FILENAME',
'lyra_20110607-000000_lev2_std.fits')])
```

Or in a slightly prettier way by viewing it as a Python dictionary:

In [7]:
```
dict(lyra.meta)
```

Out [7]:
```
{'ALGOR_V': 'EDG=2.1  BSDG=0.8',
 'BITPIX': 8,
 'DATASRC': 'Redu',
 'DATE': '2012-10-12',
 'DATE-END': '2011-06-07T23:59:59.988995',
 'DATE-OBS': '2011-06-07T00:00:00.010000',
 'EXTEND': True,
 'FILENAME': 'lyra_20110607-000000_lev2_std.fits',
 'INSTRUME': 'LYRA',
 'LEVEL': '2',
 'NAXIS': 0,
 'OBJECT': 'EUV solar irrad',
 'OBS_MODE': 'standard',
 'ORIGIN': 'ROB',
 'SIMPLE': True,
 'TELESCOP': 'PROBA2'}
```

Accessing the meta data is easy!

In [8]:
```
DATE_OBS = lyra.meta["DATE-OBS"]
```

In [9]:
```
print DATE_OBS
```

```
2011-06-07T00:00:00.010000
```

1.3 Accessing Data

The data itself is stored in the data attribute and can be quickly viewed like so.

In [10]:
```
lyra.data
```

Out [10]:

	CHANNEL1	CHANNEL2	CHANNEL3	CHANNEL4
2011-06-07 00:00:00.010000	0.006370	0.695500	0.002303	0.001039
2011-06-07 00:00:00.010000	0.006370	0.695564	0.002303	0.001038
2011-06-07 00:00:00.010000	0.006371	0.695542	0.002303	0.001038
2011-06-07 00:00:00.010000	0.006371	0.695542	0.002303	0.001038
2011-06-07 00:00:00.010000	0.006370	0.695521	0.002303	0.001038
2011-06-07 00:00:00.010000	0.006370	0.695542	0.002300	0.001038
2011-06-07 00:00:00.010000	0.006369	0.695542	0.002303	0.001036
2011-06-07 00:00:00.010000	0.006370	0.695542	0.002306	0.001038
2011-06-07 00:00:00.010000	0.006370	0.695542	0.002303	0.001041
2011-06-07 00:00:00.010000	0.006370	0.695521	0.002300	0.001039
2011-06-07 00:00:00.010000	0.006370	0.695542	0.002306	0.001038
2011-06-07 00:00:00.010000	0.006369	0.695542	0.002303	0.001038
2011-06-07 00:00:00.010000	0.006371	0.695542	0.002303	0.001037
2011-06-07 00:00:00.010000	0.006370	0.695542	0.002303	0.001039
2011-06-07 00:00:00.010000	0.006370	0.695542	0.002303	0.001037
2011-06-07 00:00:00.010000	0.006369	0.695542	0.002303	0.001036
2011-06-07 00:00:00.010000	0.006369	0.695542	0.002303	0.001038
2011-06-07 00:00:00.010000	0.006371	0.695542	0.002303	0.001039
2011-06-07 00:00:00.010000	0.006370	0.695521	0.002300	0.001041
2011-06-07 00:00:01.010000	0.006370	0.695521	0.002303	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695564	0.002306	0.001038
2011-06-07 00:00:01.010000	0.006371	0.695521	0.002300	0.001039
2011-06-07 00:00:01.010000	0.006371	0.695564	0.002303	0.001039
2011-06-07 00:00:01.010000	0.006371	0.695542	0.002303	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695521	0.002303	0.001036
2011-06-07 00:00:01.010000	0.006369	0.695542	0.002303	0.001040
2011-06-07 00:00:01.010000	0.006371	0.695564	0.002303	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695521	0.002303	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695542	0.002303	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695542	0.002303	0.001037
2011-06-07 00:00:01.010000	0.006370	0.695542	0.002303	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695542	0.002300	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695521	0.002300	0.001036
2011-06-07 00:00:01.010000	0.006370	0.695542	0.002303	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695564	0.002303	0.001037
2011-06-07 00:00:01.010000	0.006369	0.695521	0.002300	0.001038
2011-06-07 00:00:01.010000	0.006371	0.695542	0.002306	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695542	0.002303	0.001038
2011-06-07 00:00:01.010000	0.006370	0.695542	0.002303	0.001038
2011-06-07 00:00:02.010000	0.006370	0.695542	0.002303	0.001038
2011-06-07 00:00:02.010000	0.006370	0.695542	0.002303	0.001038
2011-06-07 00:00:02.010000	0.006371	0.695521	0.002303	0.001039
2011-06-07 00:00:02.010000	0.006370	0.695564	0.002303	0.001038
2011-06-07 00:00:02.010000	0.006369	0.695521	0.002303	0.001038
2011-06-07 00:00:02.010000	0.006370	0.695542	0.002303	0.001038
2011-06-07 00:00:02.010000	0.006369	0.695542	0.002303	0.001037

```
2011-06-07 00:00:02.010000  0.006369  0.695521  0.002303  0.001038
2011-06-07 00:00:02.010000  0.006370  0.695542  0.002303  0.001038
2011-06-07 00:00:02.010000  0.006369  0.695542  0.002303  0.001038
2011-06-07 00:00:02.010000  0.006371  0.695500  0.002303  0.001038
2011-06-07 00:00:02.010000  0.006370  0.695564  0.002303  0.001038
2011-06-07 00:00:02.010000  0.006370  0.695542  0.002303  0.001037
2011-06-07 00:00:02.010000  0.006369  0.695542  0.002303  0.001037
2011-06-07 00:00:02.010000  0.006370  0.695521  0.002303  0.001038
2011-06-07 00:00:02.010000  0.006369  0.695542  0.002300  0.001037
2011-06-07 00:00:02.010000  0.006370  0.695564  0.002303  0.001038
2011-06-07 00:00:02.010000  0.006371  0.695521  0.002303  0.001037
2011-06-07 00:00:02.010000  0.006370  0.695542  0.002303  0.001042
2011-06-07 00:00:02.010000  0.006371  0.695542  0.002300  0.001038
2011-06-07 00:00:03.010000  0.006370  0.695564  0.002303  0.001039
                                 ...       ...       ...       ...

[1686198 rows x 4 columns]
```

This is a pandas data frame. We can see the measurement times in the left column and the irradiance measurements for Channels 1-4 in W m^{-2} in the other columns. The measurement times are stored under the index attribute.

In [11]:
```
lyra.data.index
```

Out [11]:
```
<class 'pandas.tseries.index.DatetimeIndex'>
[2011-06-07 00:00:00.010000, ..., 2011-06-07 23:59:59.010000]
Length: 1686198, Freq: None, Timezone: None
```

And individual values can be manipulated by including their index/indices.

In [12]:
```
lyra.data.index[0]
```

Out [12]:
```
Timestamp('2011-06-07 00:00:00.010000', tz=None)
```

Meanwhile, the Channel 1 (Lyman Alpha), Channel 2 (Herzberg), Channel 3 (Aluminium) and Channel 4 (Zirconium) data can be accessed like so.

In [13]:
```
lyra.data.CHANNEL1
```

Out [13]:
```
2011-06-07 00:00:00.010000    0.006370
2011-06-07 00:00:00.010000    0.006370
2011-06-07 00:00:00.010000    0.006371
2011-06-07 00:00:00.010000    0.006371
2011-06-07 00:00:00.010000    0.006370
2011-06-07 00:00:00.010000    0.006370
2011-06-07 00:00:00.010000    0.006369
2011-06-07 00:00:00.010000    0.006370
2011-06-07 00:00:00.010000    0.006370
2011-06-07 00:00:00.010000    0.006370
2011-06-07 00:00:00.010000    0.006370
2011-06-07 00:00:00.010000    0.006369
2011-06-07 00:00:00.010000    0.006371
```

```
2011-06-07 00:00:00.010000    0.006370
2011-06-07 00:00:00.010000    0.006370
...
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006331
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006331
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006331
2011-06-07 23:59:59.010000    0.006330
2011-06-07 23:59:59.010000    0.006331
Name: CHANNEL1, Length: 1686198
```

And individual values can be manipulated by including their index/indices as with lyra.data.index.

```
In [14]: lyra.data.CHANNEL1[0]
```

```
Out [14]:
0.0063700243761586782
```

or

```
In [15]: lyra.data["CHANNEL1"][0]
```

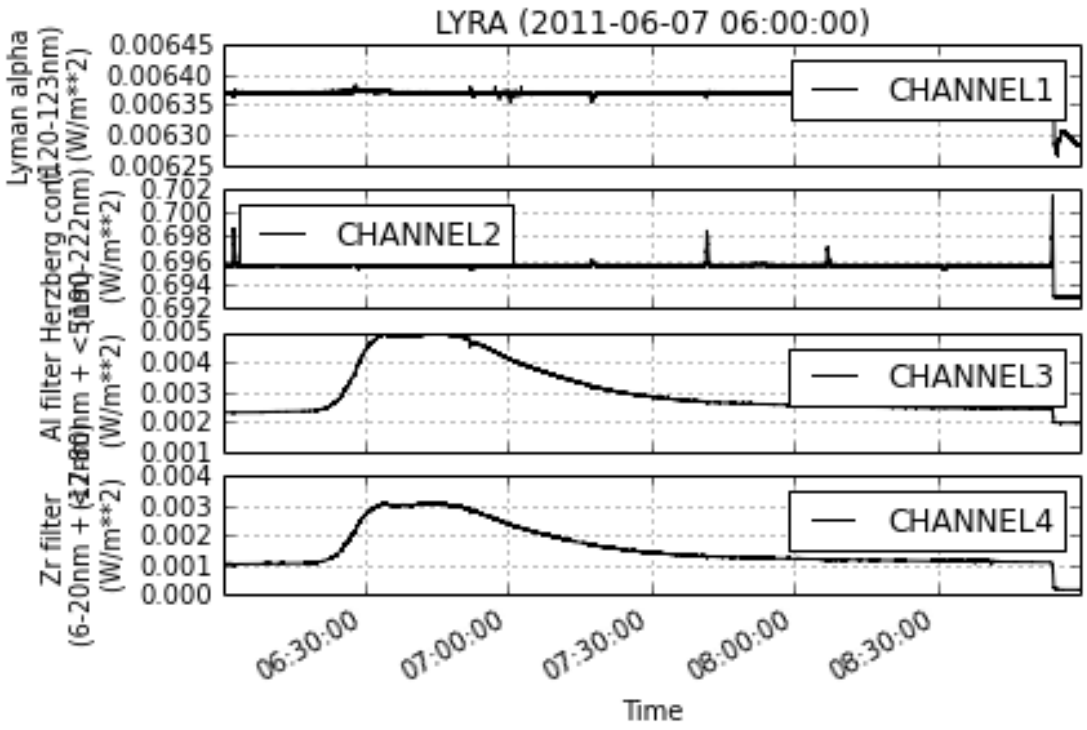
```
Out [15]:
0.0063700243761586782
```

1.4 Truncating LightCurve Object

Currently, the LYRALightCurve object can only create data frames for entire single days – corresponding to one FITS file. However once the object is created, it can be truncated to include data from a sub-time range, e.g. a flare. This is done using the truncate attribute.

```
In [16]: lyra_flare = lyra.truncate("2011-06-07 06:00", "2011-06-07 09:00")
```

```
In [17]: fig = lyra_flare.peek()
```



This should allow you to access and then manipulate LYRA data via the LYRALightCurve object. If however, you would like a more hands-on approach using the FITS files, then see the following section.

2 Manually Accessing LYRA Data

If you do not like the SunPy LYRALightCurve object, the LYRA FITS files can be accessed more directly using the AstroPy Fits module. AstroPy is a dependency for SunPy, so if you have this working, you should also have SunPy. The easiest way to get AstroPy and other useful scientific python packages is by installing the Anaconda distrubution. This is discussed on the SunPy website, http://docs.sunpy.org/en/stable/guide/installation/index.html, and for more information, see http://docs.continuum.io/anaconda/install.html.

First, download the desired LYRA FITS file from http://proba2.oma.be/data/LYRA to a given location.

```
In [18]: f = "/Users/danielr/pro/data/LYRA/fits/lyra_20110607-000000_lev2_std.fits"
```

Then, import the astropy fits module.

```
In [19]: from astropy.io import fits
```

Now open the FITS file and create a list of Header Data Units inside.

```
In [20]: hdulist = fits.open(f)
```

```
In [21]: hdulist
```

Out [21]:

```
[<astropy.io.fits.hdu.image.PrimaryHDU at 0x110a74190>,
 <astropy.io.fits.hdu.table.BinTableHDU at 0x10d55fd90>]
```

This contains two HDUs. The header information of both can be viewed like so.

In [22]:
```
hdulist[0].header
```

Out [22]:

```
SIMPLE  =                    T /Written by IDL:  Fri Oct 12 16:41:40
2012
BITPIX  =                    8 /
NAXIS   =                    0 /
EXTEND  =                    T /File contains extensions
ORIGIN  = 'ROB     '           /
TELESCOP= 'PROBA2  '           /
INSTRUME= 'LYRA    '           /
OBJECT  = 'EUV solar irrad'    /
OBS_MODE= 'standard'           /science data
DATE    = '2012-10-12'         /
DATE-OBS= '2011-06-07T00:00:00.010000' /UTC start of observation
DATE-END= '2011-06-07T23:59:59.988995' /UTC end of observation
DATASRC = 'Redu    '           /receiving ground station
LEVEL   = '2       '           /calibration level
ALGOR_V = 'EDG=2.1  BSDG=0.8'  /LYRA calibration S/W version
FILENAME= 'lyra_20110607-000000_lev2_std.fits' /name of this FITS file
```

In [23]:
```
hdulist[1].header
```

Out [23]:

```
XTENSION= 'BINTABLE'           /Written by IDL:  Fri Oct 12 16:41:40
2012
BITPIX  =                    8 /
NAXIS   =                    2 /Binary table
NAXIS1  =                   45 /Number of bytes per row
NAXIS2  =              1686198 /Number of rows
PCOUNT  =                    0 /Random parameter count
GCOUNT  =                    1 /Group count
TFIELDS =                    6 /Number of columns
EXTNAME = 'IRRAD LEVEL 2'      /name of binary table extension
TFORM1  = '1D      '           /Real*8 (double precision)
TTYPE1  = 'TIME    '           /TIME in seconds of the day
TUNIT1  = 's       '           /Units of column 1
TFORM2  = '1D      '           /Real*8 (double precision)
TTYPE2  = 'CHANNEL1'           /CHANNEL1: Lyman alpha
TUNIT2  = 'W/m**2  '           /Units of column 2
TFORM3  = '1D      '           /Real*8 (double precision)
TTYPE3  = 'CHANNEL2'           /CHANNEL2: Herzberg
TUNIT3  = 'W/m**2  '           /Units of column 3
TFORM4  = '1D      '           /Real*8 (double precision)
TTYPE4  = 'CHANNEL3'           /CHANNEL3: Aluminium filter
TUNIT4  = 'W/m**2  '           /Units of column 4
TFORM5  = '1D      '           /Real*8 (double precision)
```

```
TTYPE5  = 'CHANNEL4'           /CHANNEL4: Zirconium filter
TUNIT5  = 'W/m**2  '           /Units of column 5
TFORM6  = '5A      '           /Character string
TTYPE6  = 'WARNING '           /WARNING: quality of time and channels
TUNIT6  = '        '           /Units of column 6
```

The first HDU contains information common to all LYRA data products, while the 2nd HDU contains the data itself and specific meta data. Therefore we can access the data as numpy arrays via the data attribute of the 2nd HDU.

In [24]:
```
hdulist[1].data
```

Out [24]:
```
FITS_rec([ (0.059999999999999998, 0.0063700243761586782,
0.69550017003496967, 0.0023030435054691477, 0.0010393771404102125,
'30000'),
       (0.11, 0.0063700243764950749, 0.69556357102017263,
0.0023030435057834302, 0.0010375935158836796, '30000'),
       (0.16, 0.0063711191500763348, 0.69554243738241905,
0.0023030435060977399, 0.0010375935162705654, '30000'),
       ...,
       (86399.888995000001, 0.0063309406775694079, 0.6968830732990513,
0.0022964792278496512, 0.0010293206299999808, '30000'),
       (86399.938995000004, 0.0063298456893247082,
0.69692534892966362, 0.0022993993945162943, 0.0010315505933094617,
'30000'),
       (86399.988995000007, 0.0063306669339840919,
0.69688307333320021, 0.0022964792277739184, 0.0010284286531620998,
'30000')],
      dtype=[('TIME', '>f8'), ('CHANNEL1', '>f8'), ('CHANNEL2',
'>f8'), ('CHANNEL3', '>f8'), ('CHANNEL4', '>f8'), ('WARNING', 'S5')])
```

The final line here, starting with dtype, gives the names of the different tags of the time and various channel measurements. The time units are seconds since the start of the day, while the irradiance is W m^{-2}.

In [25]:
```
time = hdulist[1].data.TIME
```

In [26]:
```
ch4 = hdulist[1].data.CHANNEL4
```

In [27]:
```
time
```

Out [27]:
```
array([  6.00000000e-02,   1.10000000e-01,   1.60000000e-01, ...,
         8.63998890e+04,   8.63999390e+04,   8.63999890e+04])
```

In [28]:
```
ch4
```

Out [28]:
```
array([ 0.00103938,  0.00103759,  0.00103759, ...,  0.00102932,
        0.00103155,  0.00102843])
```

You can now manipulate the data as you like, for example, background subtraction, plot using matplotlib etc.

3 LYRA Artifacts and the LYRA Time Annotation Files

Having accessed the data, it is important to properly interpret it. Therefore we must know the various types of artifacts commonly found in the data, and when they occur. To easily determine which artifacts occur when, the LYRA Time Annotation Files are available at http://proba2.oma.be/data/TARDIS. This provides links to the files and explanation of what's in them.

The files themselves are SQLite3 database files and can be directly accessed in SQLite3. Alternatively, they can be accessed using Python's sqlite3 module. Software to easily extract this information and import it into Python, as well as for removing these arifacts from LYRA data have been developed by the LYRA science team and will be available to the public in the near future.