astrodata Tutorial

Release 0.1

Gemini

April 22, 2010

Contents

1 Gemini Astrodata

1.1 Introduction to Astrodata

Astrodata is a module in Python designed to handle sets of astronomical data as a single unit. Currently Astrodata only supports data in Multi-extension FITS files (MEF) coming from the Gemini instruments. The goal is to provide an interface for the non-professional programmer to easily access FITS header values and data in a way that is non instrument dependent.

1.2 Intended Audience

If you have not seen Python you could start with Example 2 for an introductory examples of Astrodata usage.

If you know IDL already, then you could quickly become familiar with Python as the syntax is very similar, with a noticeable exception that Python does not use curly braces nor 'begin' or 'end' but rather relies on blocks of statements.

1.3 Required software

To use this tutorial you need some modules that are part of the default Python distribution: the Astrodata module is available internaly via the Gemini DPD group; the Numpy module is also needed for numerical array calculation and it might be part of your Python installation.

1.4 Tutorial

1. *Example 1: ADUtoElectrons scripts under Astrodata.*
2. *Line by line explanation of Example 1*
3. *Example 2: Python and Astrodata beginners example.*
4. *Example 3: Getting information from list of GEMINI FITS file.*

1. Example 1: ADUtoElectrons scripts under Astrodata.

In order to run this example you need to have:

- Python installed in your machine
- Astrodata module. If you don't, please ask somebody from Kathleen group.
- Some GMOS data

To run the example in your UNIX shell please do:

```
# These are Unix commands
chmod u+x adu2electron.py                # Give execute mode
adu2electron.py gmos1.fits gmos2.fits    # 2 input FITS files
```

Please copy and paste this example into a file 'adu2electron.py'

```
#! /usr/bin/env python

from astrodata.AstroData import AstroData, prepOutput
from copy import deepcopy

def ADUToElectron(filelist, oprefix):
    """
      This is a function to convert the ADU counts to electrons
      by multiply the pixel values by the gain.
    """

    # Loop through the files in filelist
    for filename in filelist:

        # Open the file as an AstroData object
        adinput = AstroData(filename, mode='readonly')

        # Verify whether the data has already been converted to electrons
        if adinput.phuValue('ELECTRON') != None:
            print "WARNING: File %s has already been converted to electrons" % filename
            print "          Skipping file."
            continue

        # Prepare a new output
        #    Propagate PHU and MDF (if applicable) to output.
        #    No pixel extensions yet.
        #    Set output file name.
        #    No overwrite allowed. (default mode for prepOutput)
```

```
        #
        # prepOutput copies the adinput PHU and set the name of the new
        # file represented by adout to outputname.

        outputname = oprefix + filename
        adout = prepOutput(adinput, outputname)

        # Get the gain values to apply
        # adinput.gain() returns a list, one value for each science extension.

        gain = adinput.gain()

        # Multiply each science extension by the gain.
        # Append new extension to already prepared output.
        # Use the deepcopy function to create a true copy and ensure that
        # the original is not modified.s

        adc = deepcopy(adinput)
        for extension,g,xn in zip(adc, gain, range(len(gain))):
            extension.data = extension.data * g
            adout.append(data=extension.data, header=extension.header)

        # Update PHU with timestamps
        adout.phuSetKeyValue('ELECTRON', fits_utc(), comment='UT Modified with convertToElectrons')
        adout.phuSetKeyValue('GEM-TLM', fits_utc(), comment='UT Last modification with GEMINI')

        # Write to disk.  The filename was specified when prepOutput was called.
        adout.write()

        # Close files
        adout.close()
        adc.close()
        adinput.close()

# FITS_UTC -- local function definition
import time

def fits_utc():
    """
      Return a UTC string in FITS format: YYYY-MM-DDThh:mm:ss
    """

    gmt = time.gmtime()
    time.asctime(gmt)
    fitsT = '%d-%02d-%02dT%02d:%02d:%02d' % gmt[:6]

    return fitsT

if __name__ == "__main__":

    import optparse
    VERSION = '1.0'

    # Parse input arguments
    usage = 'usage: %prog [options] file1 .. fileN'
    p = optparse.OptionParser(usage=usage, version='v'+VERSION)
    p.add_option('--oprefix', '-p', action='store', dest='oprefix', default='elec_',
        help='Prefix for the output files')
```

```
87
88         (options, args) = p.parse_args()
89
90         ADUToElectron(args, options.oprefix)
```

2. Line by line explanation of Example3

Line 1: #! /usr/bin/env python

If you want to run the Python script as a Unix command, this statement tells the UNIX shell that indeed is a Python program and loads the interpreter.

Line 3: from astrodata.AstroData import AstroData, prepOutput

Python modules that are not part of the default installation need to be loaded using some of these constructions. 'from astrodata.AstroData import AstroData, prepOutput': Will look at the directory 'astrodata' and load the Python file 'AstroData.py' in that directory; from this file, it will make available the class Astrodata and the function prepOutput. The pathname to this directory needs to be accessible via the environment variable PYTHONPATH.

Line 6: def ADUToElectron(filelist, outprefix, verbose)

This is the function definition, its name is 'ADUToElectron'. When calling it you must give the one or more filenames in a list and the output prefix for the output files as a string, verbose is a True/ False argument.

Line 13: for filename in filelist:

The first line is a DO loop statement. It will take each element of 'filelist' and put the value in 'filename'.

Line 16: adinput = AstroData(file, mode='readonly')

This is the starting point of the 'astrodata' usage. We are opening a file and returning a list of FITS HDUs (header data units), and functions. To see a list of these functions please type 'dir adinput' in your Python shell. For help on any: 'help adinput.phuValue'

Line 19: if adinput.phuValue('ELECTRON') != None:

This function looks for the value of the PHU (primary header unit) keyword 'ELECTRON'; if there is no such keyword, it returns the value 'None'.

Line 34: adout = prepOutput(adinput, outputname)

This function returns an AstroData object with the PHU of the input file. The output filename will be 'outputname'.

Line 39: gain = ad.gain()

The fits file openned has gain information in each of the EHUs (extension header unit). This command returns a python list with those values. It assumes that the keyword name is 'GAIN'.

Line 46: adc = deepcopy(adinput)

Make a new copy of the input Astrodata object; we do not want to change the input data in memory.

Line 47: for extension,g,xn in zip(adinput, gain, range(len(gain))):

This is a compound Do loop statement of the form 'for VARS in LIST'. The 'zip' function takes as arguments one or more Python lists and returns a list of tuples, with one element at a time. For example: zip([1,2],['a','b']), returns [(1, 'a'), (2, 'b')]. The function 'range(n)' takes an integer and expands it into an increasing sequence from 0..n-1.

The VARS will then contain one set of tuples at a time.

Line 48: extension.data = extension.data * g

'extension' is a python object containing the pixel area data among other things. we just replace those values with the new ones. This is a memory operation and does not affect the input file.

Line 49: adout.append(data=extension.data, header=extension.header)

We now append a new FITS extension to the output file with its data and header.

**Line 52: adout.phuSetKeyValue('ELECTRON', fits_utc(), **

comment='UT Modified with convertToElectrons')

Appends the indicated keyword to the PHU; its value is the value of the function 'fits_utc()' defined at the end of the Example1. Notice that Python allows for a statement to be broken in different lines; mostly a comma is a good line separator.

Line 56: adout.write()

Writes to disk the updated output AstroData object. The filename was already defined in 'prepOutput()'

3. Example 2: Python and Astrodata beginners example.

```
# Starting PYTHON in unix and MacOS.
#
# Astrodata module needs to be visible to Python via the
# unix environment variable PYTHONPATH. Please append a
# pathname to this variable where Astrodata is located in
# your machine.
#
# Example: setenv PYTHONPATH ${PYTHONPATH}:/home/user/soft/astrodata
#
# Now start 'ipython', (a better python shell) and try these commands.

ipython


from astrodata.AstroData import AstroData    # Load Astrodata module

import numpy as np              # Load the numpy module and define an alias 'np'

file = 'N20020214S060.fits'     # Define a variable 'file' with a string.

ad = AstroData(file)            # Open the FITS file using Astrodata
                                # 'ad' is the Astrodata object. 'ad' is just
                                # a variable name.

ad.info()                       # Get FITS file information. This notation is python
                                # syntax stating that one of the function of the object
                                # 'ad' is info(). For a list of all 'ad' functions i
                                # please type 'dir ad'.

print ad.info()                 # Same as above. 'print' is optional is most cases using
                                # the python interpreter.

help ad.info                    # Get help on the 'info' function
help(ad.info)                   # (Same) Get help on the 'info' function
                                # The parenthesis are mostly optional using the
                                # python interpreter. If you are usign PYRAF,
```

```
                                  # these are required.
# Astrodata descriptors

ad.getTypes()                     # Will list the different Astrodata types that this
                                  # Gemini FITS file has. As the data goes through different
                                  # processing stages, these values will change.

ad.isType('GMOS_IMAGE')           # Boolean function. True is the data file is of type
                                  # 'GMOS_IMAGE'


# FITS pixel data with Astrodata

data = ad[0].data                 # Get the first extension pixel data and assign it to
                                  # the variable 'data' containing the pixels array.
                                  # 'ad' is a Python list pointing to FITS extension
                                  # header and data; hence 'ad[0]' points to the first
                                  # extension, 'ad[1]' to the second, etc.

dim = np.shape(data)              # Get array dimensionality. Parenthesis are not needed.
                                  # Store the values in 'dim'. Here parenthesis are necessary
                                  # when you are assigning to a variable.

dim                               # show the values. You can also use 'print dim'

np.median(data)                   # Calculates the median od the pixels in the 'data' array.
                                  # 'np.median' means 'get the median function from the
                                  # numpy (np) module and use it with the data array'

np.median(data[100,:])            # Calculates the median of row 100.
                                  # In Python the array order is [y,x] (IDL is [x,y]).
                                  # ':' means all the pixels in the row

np.std(data)                      # Get the standard deviation
help np.std                       # see what the help section says about it

np.amax(data); np.amin(data)      # 2 commands in one line (separator is ';')

dir ad                            # See all the functions available with the
                                  # object 'ad' associated to the FITS file.

# FITS HEADER with Astrodata

hdr = ad[0].header                # Get 1st extension header
hdr.items()                       # list keywords with their values
hdr['crpix1']                     # print keyword value

phu = ad.getPHUHeader()           # Get Primary Header Unit (note the (); it's a function)
phu.items()
phu['datalab']
```

4. Example 3: Getting information from list of GEMINI FITS file

Example 2 is an interactive Python session with separate commands that give you results right at the terminal.

Example 3 is a Python script written into a text file which you can run at the Unix prompt or as a single command in the Python Shell.

Copy and paste the source below is a Python called 'list_FITS_info.py'

Running from Unix

Notice that this file contains one function called 'list_info' and that the first line is '#! /usr/bin/env python' telling the UNIX system to use the Python interpreter to execute the rest of the script. This happens when you type 'list_FITS_info.py' at the unix prompt.

The file needs to have executable mode:

```
chmod u+x list_FITS_info.py
```

To run 'list_FITS_info.py' simply type:

```
list_FITS_info.py

or equivalently

python list_FITS_info.py    # you do not need execute mode in this case
```

Running from the Python Shell

- Assuming you are where 'list_FITS_info.py' is located.
- Tell Python that you have a file (list_FITS_info.py) with a function 'list_info':

```
from list_FITS_info import list_info

# now run the script:

list_info()             # The scripts does not have arguments, but you need
                        # the parentheses.
```

list_FITS_info.py. Copy and paste into a file called 'list_FITS_info.py':

```
#! /usr/bin/env python

from astrodata.AstroData import AstroData
import glob

def list_info():

    # Generates a list of FITS files in directory '/data2/rtf_data'
    filelist = glob.glob('/data2/rtf_data/*.fits')

    # Placeholder information for the fields to be output
    print 'nsciext: camera: exptime: filtername: instrument: object: ut'

    for file in filelist:          # Loop over all the items in filelist.

        ad = AstroData(file)       # Open the current file

        # Get information for files that are type 'NIRI' and 'GMOS' only.

        if not ad.isType('NIRI') and not ad.isType('GMOS'):
            continue               # Skip to next file in filelist is not
                                   # NIRI nor GMOS

        print '****** ',file
```

```
        nxts = ad.countExts('SCI')       # Get number of FITS Science extensions

        print ad.getTypes()              # show the Astrodata types associated
                                         # with this FITS file and its current
                                         # processing state.

        print ad.camera()                # Show the value of the PHU keyword 'CAMERA'

        print ad.exptime()               # This is the actual EXPTIME*NCOADDS value

        print ad.filtername()            # Filtername is a unique string discribing
                                         # the filter for the instrument.

        print ad.instrument(), ad.object(), ad.utdate()    # Header keyword values

        if ad.isType('GMOS'):
            if nxts > 0:
                print "GAIN:", ad.gain()    # print a list of gains
            # Based on the value of the PHU keywords MASKTYPE, MASKNAME
            # and GRATING, the followinf values can be inferred:
            # IMAGE, IFU, LONGSLIST or  MOS.
            print "OBSMODE:", ad.obsmode(),'\n'

        if ad.isType('NIRI'):
            print 'NIRI FILTER header keywords:',\
            # this next form is a Python 'list comprehension' syntax. It is
            # a compressed 'for loop'.
            [(ff+'='+ad.getPHU()[ff]) for ff in ad.rePHUKeys('FILTER\w*')]

            print 'From niri.descriptor:', ad.filtername()
            print 'Non Linear value:', ad.nonlinear()

if __name__ == "__main__":
   list_info()
```