Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

Demo of numarray, PyFITS, and related software

Jin-chung Hsu and Philip E. Hodge

Space Telescope Science Institute, Baltimore, MD 21218

Abstract. PyFITS is a Python module developed for FITS file I/O. We demonstrated its use for FITS images and tables, illustrating the PyFITS classes and methods, as well as the array manipulation capabilities of numarray. PyFITS is convenient for interactive use, and we also showed two utility programs, **fitsdiff** and **readgeis**, as examples of its use in astronomical applications. The task **fitsdiff** compares two FITS files and reports their differences, **readgeis** reads the STScI-style GEIS format files and converts them to FITS files or FITS objects. These two Python modules were showcased not only because they are useful astronomical tools but to demonstrate the ease of writing such applications using PyFITS and numarray. PyFITS can also make use of memory mapping, which significantly enhances its performance on large FITS files, both images and tables. At STScI, we are also applying numarray and PyFITS for larger projects such as pydrizzle and the pipeline software for the new HST instrument COS (Cosmic Origins Spectrograph).

1. Introduction

The Python package numarray is an efficient array handling tool (it is a replacement for the Numeric Python extension). In addition to the existing Numeric functionalities for numeric arrays, arrays of character strings can be created and manipulated, and tables can be represented using record arrays which are 1-D arrays of structures with mixed data types within a row but are homogeneous within each column. The contents of a record array can be accessed by row or by column, or both at the same time. A column is a numarray (or numarray.strings) object, with attributes that include an offset from one element to another, which allows accessing the column data without copying from the table to a temporary array.

PyFITS is a Python module for working with FITS files. For an image, the data block is a numarray object. For a table, the data block is a record array. Images and tables may be read, updated in-place, or they may be created from scratch. Header keywords may be read, modified, deleted, or inserted at any location.

Memory mapping is supported in PyFITS to improve performance for large files. If the native byte order differs from that of a FITS file (big-endian), byte swapping is done on-the-fly within numarray when accessing data. This avoids the need for temporary storage space, and it can save time if only a portion of

the data will be read or modified. Verification of objects to adhere to the FITS standards can also be performed automatically or manually.

If the data in the FITS file is scaled, i.e. BSCALE $\neq 1$ or BZERO $\neq 0$, PyFITS handles this in a transparent way so the user only needs to interact with the scaled objects. On the other hand, this usually entails extra memory space for the scaled arrays; this may be of consideration for very large data files.

2. PyFITS Examples

In order to access an existing FITS file, we need to use the PyFITS **open** function which will return a Python list-like object (called HDUList object) which can only contain FITS HDU (header-data unit) objects as its elements.

```
>>> import numarray, pyfits
>>> fd = pyfits.open ("abc.fits", "update", memmap=1)
```

The **info** method is useful to find out what is in the HDUList:

```
>>> fd.info()
Filename: (No file associated with this HDUList)
No.   Name     Type     Cards Dimensions Format
0   PRIMARY PrimaryHDU      4 ()
1   sci     ImageHDU       61 (512, 512) Int16
2   eng     BinTableHDU    16 100R x 4C  [1D, 1J, 1J, 1I]
```

Thus the primary HDU is fd[0], the first extension HDU is fd[1], etc. The two most important attributes of each HDU object are **header** and **data**. Header keyword values can be accessed by name or index, i.e. the header object behaves like a list as well as a dictionary.

```
>>> fd[0].header['detector']
'CCD1'
>>> fd[0].header[0]
TRUE

# update an existing keyword's value and add a new keyword
>>> fd[0].header['targname']='new_name'
>>> fd[0].header.update('newkey', 42)

# add a HISTORY, and delete a keyword
>>> fd[0].header.add_history('test')
>>> del fd[0].header['newkey']
```

To access the image data is straightforward, since the data attribute of an image HDU is just a numarray object. We can use the familiar slicing and striding notations, as well as all the mathematical operations provided in numarray.

```
>>> x=f[1].data[:2,:2]; print x
[[313 312]
 [314 314]]
```

```
>>> f[1].data[:2,:2] > 313
array([[0, 0],
       [1, 1]], type=Bool)
```

Table data can be accessed by rows, using slicing or indexing,

```
>>> print fd[2].data[:3]
RecArray[
(6056.1279296875, 3626, 470, 0),
(6056.1279296875, 2735, 472, 0),
(6056.1279296875, 8410, 470, 0)
]
>>> print fd[2].data[0]
(6056.1279296875, 3626, 470, 0)
```

or by columns, by referencing the column name or index, via the **field** method.

```
# access the x and y columns
>>> x = fd[2].data.field ("x")
>>> y = fd[2].data.field ("y")

# apply a linear mapping to x and y
# (x and y are numarray objects)
>>> x = xscale * x + xzero
>>> y = yscale * y + yzero

# flag out-of-bounds in the data quality column
>>> dq = fd[2].data.field ("dq")
>>> dq |= where (logical_or (x < 0, x >= nx), 1, 0)
>>> dq |= where (logical_or (y > 0, y >= ny), 1, 0)
>>> fd.close()

# create a table from scratch
>>> col = []
>>> col.append (pyfits.Column (name="w", format="5A", \
...     array=numarray.strings.array (["one", "two", \
...     "three", "four"])))
>>> col.append (pyfits.Column (name="x", format="3E", \
...     array=array (([1.,2.,3.],[7.,8.,9.],[13.,14.,15.]))))
>>> col.append (pyfits.Column (name="y", format="J", \
...     array=array ([1,2,3,4])))
>>> col.append (pyfits.Column (name="z", format="C", \
...     array=array ([1+2j,3+4j,5+6j])))
>>> tb_hdu = pyfits.new_table (col)
>>> fd = pyfits.HDUList (tb_hdu)

# check for problems
>>> fd.verify()
Output verification result:
HDUList's 0th element is not a primary HDU.
```

```
# insert missing primary HDU
>>> fd.insert (0, pyfits.PrimaryHDU())
>>> fd.info()
Filename: (No file associated with this HDUList)
No.   Name     Type     Cards Dimensions Format
0   PRIMARY PrimaryHDU      4 ()
1   None    BinTableHDU    16 4R x 4C    ['a5','3f4','i4','c8']

# write to a disk file
>>> fd.writeto ("example.fits")
```

For more details please visit:

```
http://www.stsci.edu/resources/software_hardware/numarray
http://www.stsci.edu/resources/software_hardware/pyfits
```

3. Some applications which use PyFITS or/and numarray

```
PyTables  -- a hierarchical database package designed to
             efficently manage very large amounts of data
PyDrizzle -- a Python-based interface for combining
             overlapping images
readgeis  -- converts STScI-format images to FITS
fitsdiff  -- compares FITS images and tables
calcos    -- pipeline calibration program for COS
```

The two applications, **readgeis** and **fitsdiff**, are examples of applying numarray and PyFITS to do useful work. Both are entirely written in Python and can be run as a shell command or loaded in as a module inside Python. The task **fitsdiff** is a tool to compare two FITS files and generate customizable reports. Users can, for example, specify the relative difference level to be flagged for floating point numbers and exclude selected keywords or columns for comparison.

The task **readgeis** is a tool to read the GEIS file format used by some older HST instruments. The tool will read the GEIS file's contents into PyFITS's HDUList structure. Thus the header keywords will be in the primary HDU, the group parameters and data in each group will be in the extension HDU. Users can then either write it out as a FITS file or do mathematical operations with the data in the images or the headers.