The STAK Project

Release 1.0.0

Sara Ogaz, Justin Ely

Nov 05, 2020

CONTENTS

Below you will find Jupyter Notebook tutorials organized by usage/subject. Each notebook is grouped by one library/sub-library of IRAF tasks, which corresponds to the title of the notebook. Putting together these tutorial notebooks is an ongoing task, and contributions are welcome.

If you are new to Python or Astropy, we recommend starting with the introduction page to see some Python and Astropy information that will be used extensively in these tutorials.

For questions or comments please see our github or you can visit the STScI help page.

CHAPTER

ONE

PYTHON INTRODUCTION

This notebook will introduce new Python and Astropy users to various aspects of the Python astronomy workflow that may help them to take better advantage of the rest of the STAK notebooks. If you are a new Python or Astropy user, we highly recommend reading through all sections of this notebook. We will discuss:

- *Workflow Philosophy, the difference in workflow between IRAF and Python*.
- *FITS File I/O, opening and updating FITS files in Python*.
- *IRAF/IDL to Python gotchas*.
- *Helpful links and additional resources*.

1.1 Workflow Philosophy

The traditional IRAF workflow consists of individual tasks that take a FITS file as input, run a data processing algorithm on the input file, and return an updated output file for the user. The workflow in Python is very different. Like most interpreted languages, data manipulation is more a flow of operations expressed in regular code than individual tasks that operate on files.

For example, instead of providing an input FITS file to a task and getting an updated output file, for many astronomy Python functions the user will open the FITS file themselves, access the file object, and continue with any analysis or data processing from there. Once the data manipulation is done (which could be a long sequence of manipulations), if the user would like to update their FITS files they actively save the changes to the file. This could include saving out intermediate steps to new files, if the user so desires. While this may take a few more lines of code to open and close files, you gain the major advantage of flexibility, customization and freedom, as well as potentially more efficient I/O. Oftentimes the increase in the number of lines of code allows the user to be more explicit about what the code is doing, instead of relying on complex parameter files and black box code. If a user finds themselves running the same code numerous time and would find a one line call useful, they can always put their workflow into a function in a Python file and it will be ready to import and use in any local Python session.

Particularly useful for this kind of scientific workflow is the Jupyter Notebook. This tool allows users to run fully interactive Python sessions with the ability to save the code that was run, its screen output, the order it was run in, and other helpful tools. If you are new to Python, we highly recommend taking advantage of this tool. All STAK notebooks were written in Jupyter Notebooks. This means the user can downloads these notebooks and run all code examples locally, adding additional code to use their data in the provided examples. For a more in-depth example of how to effectively use Python and the Jupyter Notebooks for data analysis, there are some good video tutorials by Jake VanderPlas here.

1.2 FITS File I/O

Utilities to open, edit, and save FITS files are contained in the Astropy package. Although there are many documentation resources for this package (which we will reference below in the *links and resources section*), it can be a bit overwhelming for new users. To help new users adjust to a more Pythonic workflow, some of the examples shown in the STAK notebooks do not show the opening and closing of the FITS file. Instead, we show the basic calls to do this below.

Before we show any code, we will go over a basic explanation of the Astropy classes and object types for FITS files. The highest level class for a FITS file is the HDUList. This is the object type you get when you first open a FITS file in Astropy. The HDUList contains a sequence of HDU (Header Data Unit) objects. There are three common flavors of HDU object, a PrimaryHDU, an ImageHDU and a BinTableHDU. As you would expect, the PrimaryHDU object contains the primary header, the ImageHDU object contains an image array and its associated header, and a BinTableHDU object contains a binary table and its associated header.

The next level of objects are the individual header and data objects. Headers are stored in a Header object. Images are stored in a Numpy array object. Binary tables are stored in a record array.

Now that we've covered the object types, we will show with code snippets how to extract and work with these objects using a FITS file containing image data. For more information on table data see the Astropy FITS Table tutorial. For more detailed coverage of the available tools in the Astropy FITS module see the narrative documentation page.

```
from astropy.io import fits
```

```
# To pull our sample data file we will use an astroquery call
from astroquery.mast import Observations
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
INFO: Found cached file ./mastDownload/HST/iczgs3ygq/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Assign filename to the variable test_file
test_file = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
```

Below we will open the FITS file. You can open the file in various modes, for this example we will open in update mode. The default mode is read only.

```
# Open the FITS file with Astropy
HDUList_object = fits.open(test_file, mode='update')
```

Next we will show the info print out for this HDUList object using the info() method. Notice the No. and Type columns. These will be useful for indexing the HDUList.

```
# HDUList info call
HDUList_object.info()
```

```
Filename: ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     265   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
  3  DQ            1 ImageHDU        43   (1014, 1014)   int16
  4  SAMP          1 ImageHDU        37   (1014, 1014)   int16
```

(continues on next page)

(continued from previous page)

```
  5  TIME          1 ImageHDU        37   (1014, 1014)   float32
  6  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
```

Now we will extract the primary header into the variable `primary_header`

```
# Extract primary header
primary_header = HDUList_object[0].header

# Index header object with keyword name and print value
print(primary_header['FILENAME'])
```

```
iczgs3ygq_flt.fits
```

Next we extract the image data into a variable called `image_data` from the first image extension. We will index this using the index number from the No. column returned by `info()`. This variable is a numpy array object and the object that allows you to directly interact with the image data. For more information on indexing here is a useful Numpy documentation page.

```
# Extract image data from the first extension
image_data = HDUList_object[1].data
print(image_data)
```

```
[[  0.88747919   0.83535039   0.80967814 ...,   3.19881892   4.66315889
   12.94333744]
 [  0.94745165   0.80834782   0.76161045 ...,   0.91167408   3.91721344
    2.38371158]
 [  0.86024958   0.86270761   0.85969168 ...,   2.71301699  -4.11855459
    2.52296972]
 ...,
 [ 33.32045746  23.79335022   4.87152386 ...,  22.54588509  21.88571739
   23.2428627 ]
 [ 47.97618103   1.16626728  13.08955574 ...,  12.46915627  21.59257698
   16.61116219]
 [ 30.99951744  29.15618515  46.40042877 ...,   0.           9.47169876
   20.67056084]]
```

We now have two options for saving out the FITS information.

We can save it out to the original file by using our `HDUList` file object and the `close` argument. If the file was opened using the update mode, this will flush (write) the file changes. If the file was opened in the default readonly mode, it will **not** be updated when closed.

We can also use the `writeto` method to save the `HDUObject` to a new file. `writeto` will close the new file for you.

The `flush` method will also save to the original file. In this case, the original file handling object will still need to be closed at some point in the session.

No matter which mode you used to open a FITS file, you should still call the close method to close any open FITS file.

```
# Save using the writeto method to a new file, writeto will close the new file for you
HDUList_object.writeto("wfc3data_new.fits")

# Save using the writeto method, overwriting the original file
HDUList_object.flush()
```

```
# Save to same file using close
# We show this last because we need to close the original copy of the file we opened,␣
↪even after using a writeto
HDUList_object.close()
```

1.3 IRAF/IDL to Python Gotchas

There are some important differences in syntax between IRAF, IDL, and Python that new users should keep in mind. For more in depth information about indexing and slicing `Numpy` arrays see their indexing documentation here.

1.3.1 x versus y

When working with images (2-dimensional) arrays, IRAF and IDL both have the index order `[x, y]`. In Python's `Numpy` package, the order is reversed, `[y, x]`.

1.3.2 index 0

IRAF indexes begin at 1 whereas Python and IDL both index arrays starting at zero. So to pull out the first element of a 1-dimensional array you would use `array[0]`. To pull out the lower left corner of a 2-dimensional array you would use `array[0,0]`.

1.3.3 slicing

Slicing in IRAF and IDL is inclusive for the right side of the slice. In Python the right side of the slice is exclusive. For example, if you end a slice with the 4th index, `array[0:4]`, the fourth index element (actually the 5th element in the array since index begins at 0) will **not** be included in the slice.

1.3.4 matplotlib origin

The default origin location for `matplotlib` plots (a common Python plotting library) will be in the upper-left. To change this to the lower left (common for images) you can use the `origin=lower` parameter in the `imshow` call as follows: `plt.imshow(..., origin='lower')`.

1.3.5 close your files!

Almost any file handling object you open in Python (and this included a FITS file opened with the Astropy open function!) will need to be closed in your Python session with the appropriate close command. See above section for examples.

1.4 Links and Resources

1.4.1 Astropy

Main user documentation page: http://docs.astropy.org/en/stable/

Main FITS page: http://docs.astropy.org/en/stable/io/fits/index.html

Tutorials: http://www.astropy.org/astropy-tutorials/

1.4.2 Scipy and Numpy

Main documentation pages: https://docs.scipy.org/doc/

Numpy indexing guide: https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.indexing.html

1.4.3 CCDProc

Main documentation page: http://ccdproc.readthedocs.io/en/latest/

1.4.4 Matplotlib

Main documentation page: https://matplotlib.org/

1.4.5 Ginga

Main documentation page: http://ginga.readthedocs.io/en/latest/

1.4.6 Astroconda

Main documentation page: http://astroconda.readthedocs.io/en/latest/

CHAPTER

TWO

CONTENTS

Image Manipulation:

2.1 images.imfilter

The images.imfilter IRAF package provides an assortment of image filtering and convolution tasks.

2.1.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Python replacements for the images.imfilter tasks can be found in the Astropy and Scipy packages. Astropy convolution offers two convolution options, `convolve()` is better for small kernels, and `convolve_fft()` is better for larger kernels, please see the Astropy convolution doc page and Astropy Convolution How to for more details. For this notebook, we will use `convolve`. Check out the list of kernels and filters avaialble for Astropy, and Scipy

Although `astropy.convolution` is built on `scipy`, it offers several advantages: * can handle NaN values * improved options for boundaries * provided built in kernels

So when possible, we will be using `astropy.convolution` functions in this notebook. The ability to handle NaN values allows us to replicate the behavior of the zloreject/zhireject parameter in the imfilter package. See the *rmedian* entry for an example.

You can select from the following boundary rules in `astropy.convolution`: * none * fill * wrap * extend

You can select from the following boundary rules in `scipy.ndimage.convolution`: * reflect * constant * nearest * mirror * wrap

Below we change the matplotlib colormap to `viridis`. This is temporarily changing the colormap setting in the matplotlib rc file.

Important Note to Users: There are some differences in algorithms between some of the IRAF and Python Interpolations. Proceed with care if you are comparing prior IRAF results to Python results. For more details on this issue see the filed Github issue.

Contents:

- *laplace*
- *median-rmedian*
- *mode-rmode*

```
# Temporarily change default colormap to viridis
import matplotlib.pyplot as plt
plt.rcParams['image.cmap'] = 'viridis'
```

2.1.2 boxcar

Please review the *Notes* section above before running any examples in this notebook

The boxcar convolution does a boxcar smoothing with a given box size, and applies this running average to an array. Here we show a 2-D example using `Box2DKernel`, which is convinient for square box sizes.

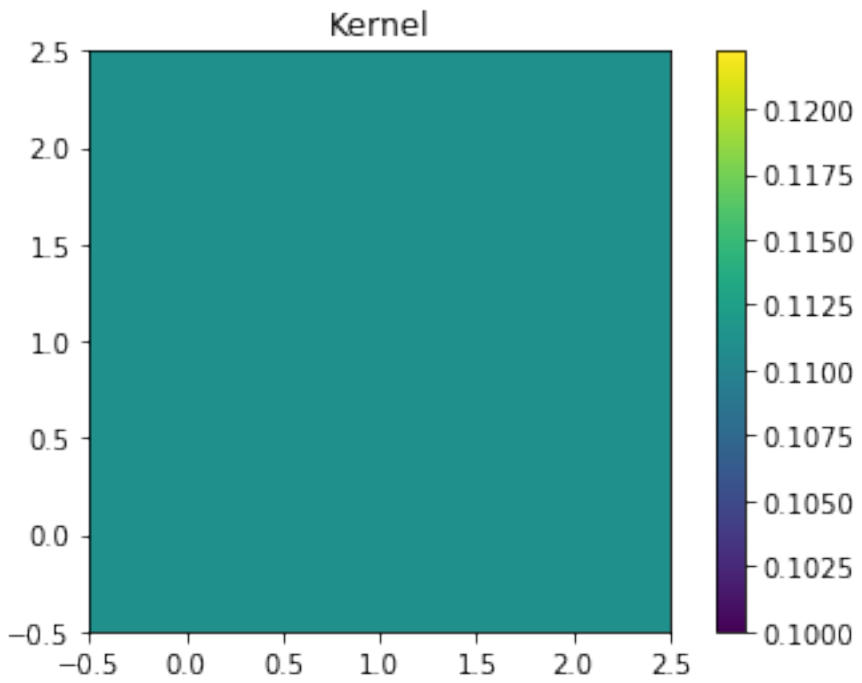
```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astropy.convolution import convolve as ap_convolve
from astropy.convolution import Box2DKernel
from astroquery.mast import Observations

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

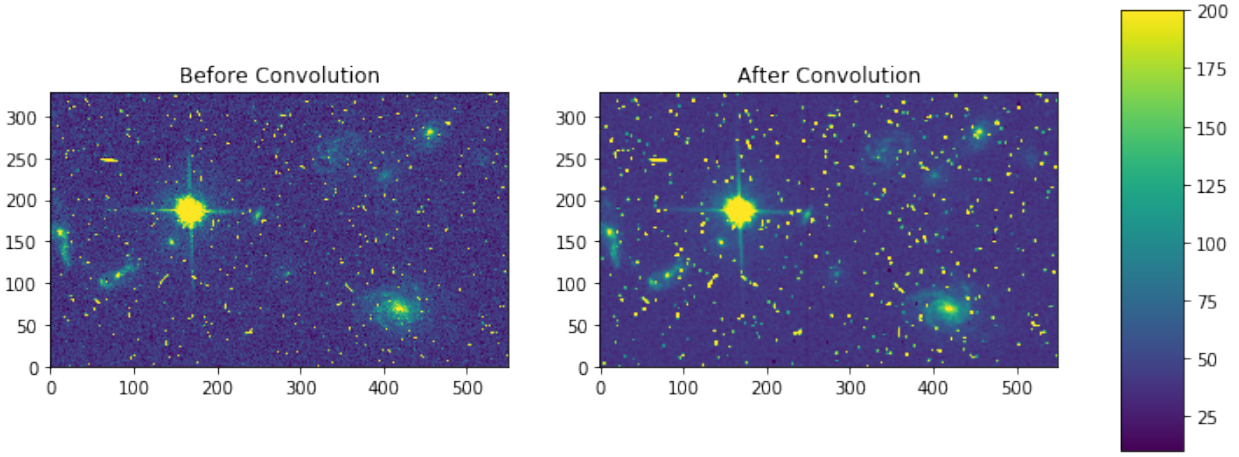
```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
# grab subsection of fits images
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
sci1 = fits.getdata(test_data,ext=1)
my_arr = sci1[700:1030,2250:2800]

# setup our kernel
box_kernel = Box2DKernel(3)
# perform convolution
result = ap_convolve(my_arr, box_kernel, normalize_kernel=True)
```

```
plt.imshow(box_kernel, interpolation='none', origin='lower')
plt.title('Kernel')
plt.colorbar()
plt.show()
```




```
fig, axes = plt.subplots(nrows=1, ncols=2)
pmin,pmax = 10, 200
a = axes[0].imshow(my_arr,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[0].set_title('Before Convolution')
a = axes[1].imshow(result,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[1].set_title('After Convolution')

fig.subplots_adjust(right = 0.8,left=0)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(a, cax=cbar_ax)
fig.set_size_inches(10,5)
plt.show()
```



2.1.3 convolve

Please review the *Notes* section above before running any examples in this notebook

The convolve task allows you to convolve your data array with a kernel of your own creation. Here we show a simple example of a rectangular kernel applied to a 10 by 10 array using the `astropy.convolution.convolve` function

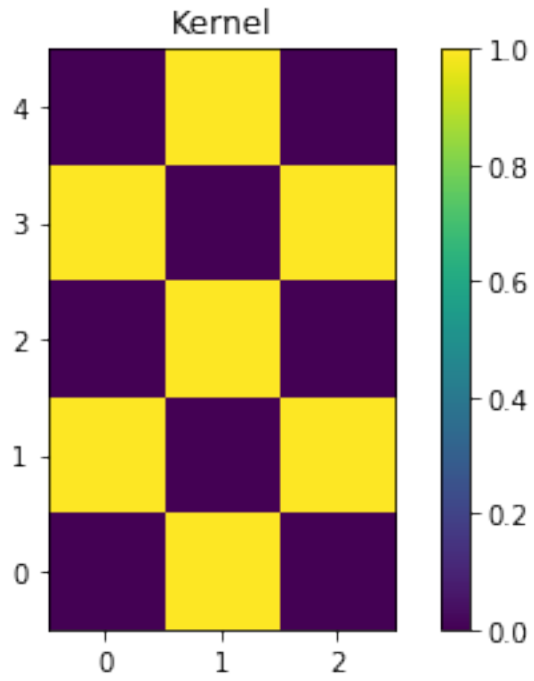
```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astropy.convolution import convolve as ap_convolve
from astroquery.mast import Observations

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

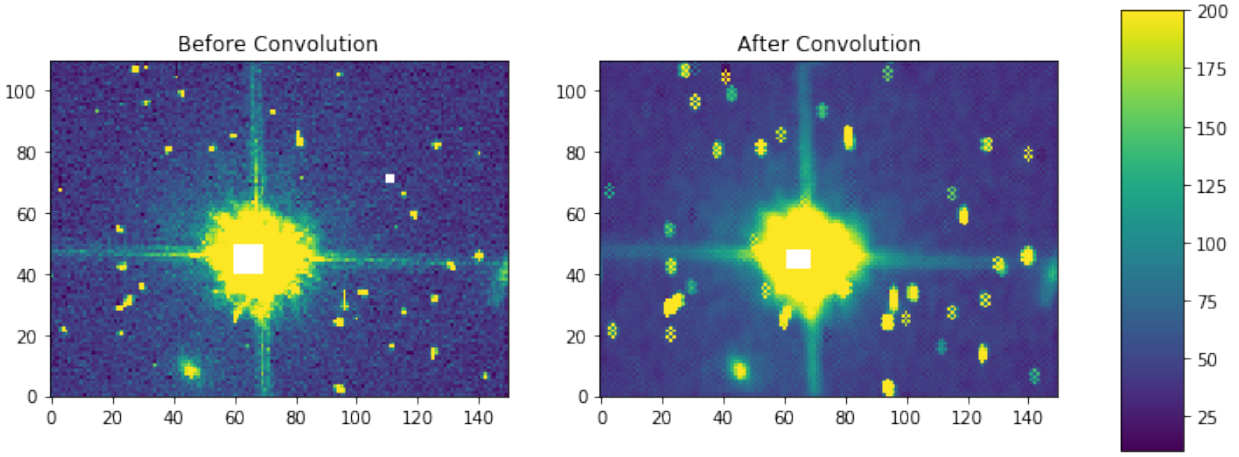
```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
# grab subsection of fits images
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
sci1 = fits.getdata(test_data,ext=1)
my_arr = sci1[840:950,2350:2500]

# add nan's to test array
my_arr[40:50,60:70] = np.nan
my_arr[70:73,110:113] = np.nan

# setup our custom kernel
my_kernel = [[0,1,0],[1,0,1],[0,1,0],[1,0,1],[0,1,0]]
# perform convolution
result = ap_convolve(my_arr, my_kernel, normalize_kernel=True, boundary='wrap')
```

```
plt.imshow(my_kernel, interpolation='none', origin='lower')
plt.title('Kernel')
plt.colorbar()
plt.show()
```




```
fig, axes = plt.subplots(nrows=1, ncols=2)
pmin,pmax = 10, 200
a = axes[0].imshow(my_arr,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[0].set_title('Before Convolution')
a = axes[1].imshow(result,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[1].set_title('After Convolution')

fig.subplots_adjust(right = 0.8,left=0)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(a, cax=cbar_ax)
fig.set_size_inches(10,5)
plt.show()
```

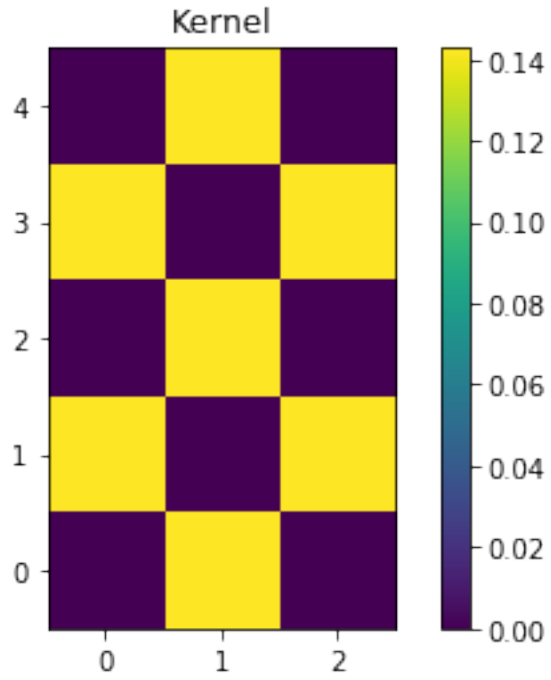


Here is an example using masking with `scipy.convolve`

```
# Standard Imports
import numpy as np
from scipy.ndimage import convolve as sp_convolve

# Astronomy Specific Imports
from astropy.io import fits

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# grab subsection of fits images
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
sci1 = fits.getdata(test_data,ext=1)
my_arr = sci1[700:1030,2250:2800]

# setup our custom kernel
my_kernel = np.array([[0,1,0],[1,0,1],[0,1,0],[1,0,1],[0,1,0]]) * (1/7.0)
# perform convolution
result = sp_convolve(my_arr, my_kernel, mode='wrap')
```

```
plt.imshow(my_kernel, interpolation='none', origin='lower')
plt.title('Kernel')
plt.colorbar()
plt.show()
```

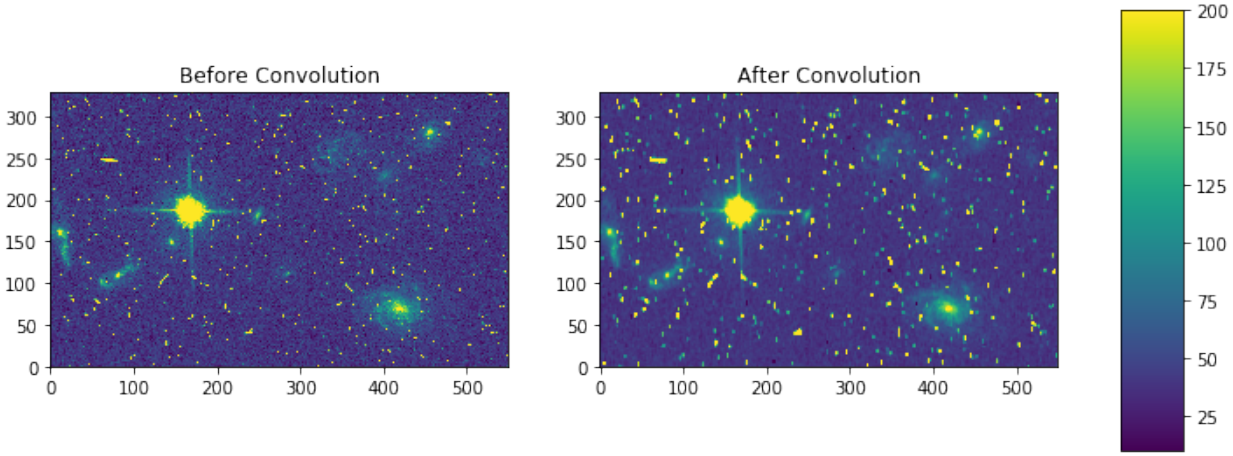



```
fig, axes = plt.subplots(nrows=1, ncols=2)
pmin,pmax = 10, 200
a = axes[0].imshow(my_arr,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[0].set_title('Before Convolution')
a = axes[1].imshow(result,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
```

(continues on next page)

(continued from previous page)

```
axes[1].set_title('After Convolution')

fig.subplots_adjust(right = 0.8,left=0)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(a, cax=cbar_ax)
fig.set_size_inches(10,5)
plt.show()
```



2.1.4 gauss

Please review the *Notes* section above before running any examples in this notebook

The gaussian kernel convolution applies a gaussian function convolution to your data array. The Gaussian2DKernel size is defined slightly differently from the IRAF version.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astropy.convolution import convolve as ap_convolve
from astropy.convolution import Gaussian2DKernel
from astroquery.mast import Observations

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

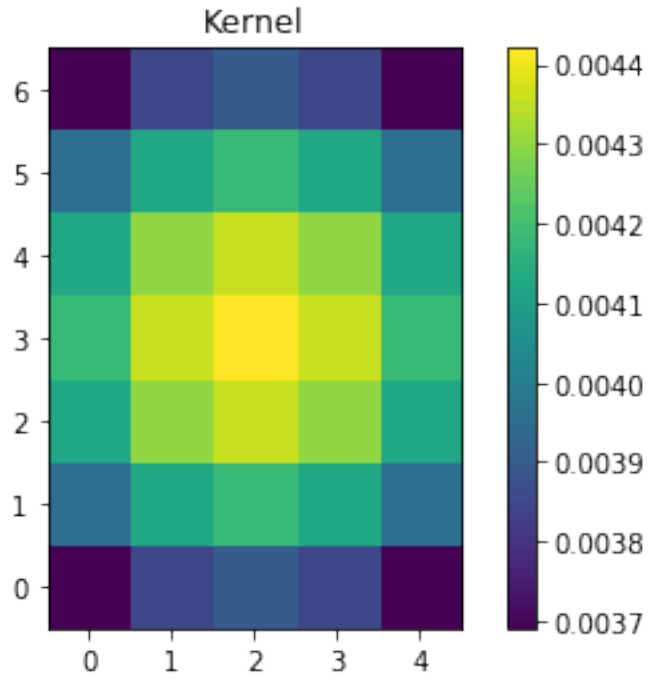
```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

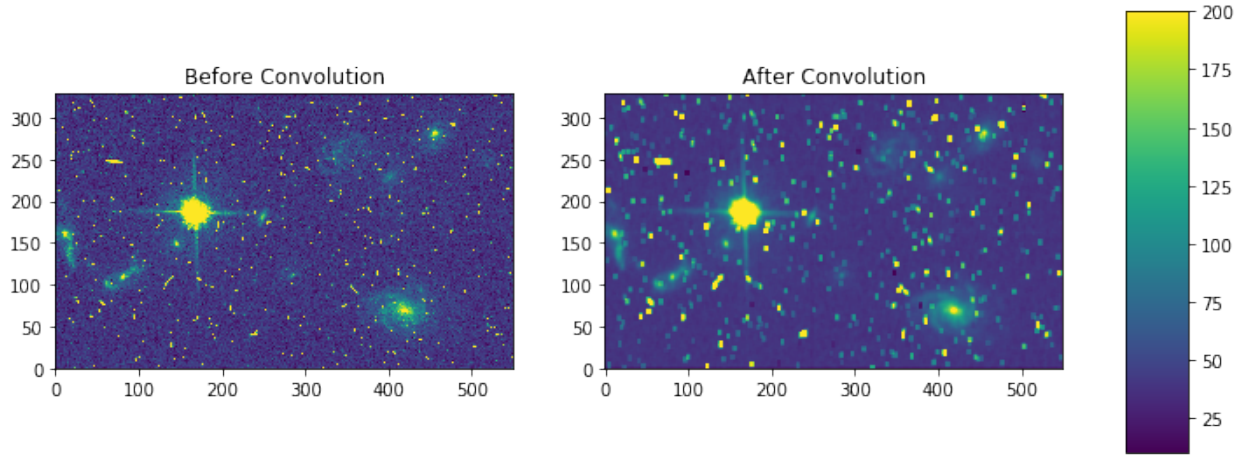
```
# grab subsection of fits images
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
sci1 = fits.getdata(test_data,ext=1)
my_arr = sci1[700:1030,2250:2800]

# setup our kernel, with 6 sigma and a 3 in x by 5 in y size
gauss_kernel = Gaussian2DKernel(6, x_size=5, y_size=7)
# perform convolution
result = ap_convolve(my_arr, gauss_kernel, normalize_kernel=True)

gauss_kernel
```

```
<astropy.convolution.kernels.Gaussian2DKernel at 0x18255efe80>
```

```
plt.imshow(gauss_kernel, interpolation='none', origin='lower')
plt.title('Kernel')
plt.colorbar()
plt.show()
```




```
fig, axes = plt.subplots(nrows=1, ncols=2)
pmin,pmax = 10, 200
a = axes[0].imshow(my_arr,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[0].set_title('Before Convolution')
a = axes[1].imshow(result,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[1].set_title('After Convolution')

fig.subplots_adjust(right = 0.8,left=0)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(a, cax=cbar_ax)
fig.set_size_inches(10,5)
plt.show()
```



2.1.5 laplace

Please review the *Notes* section above before running any examples in this notebook

The laplace task runs a image convolution using a laplacian filter with a subset of footprints. For the `scipy.ndimage.filter.laplace` function we will be using, you can feed any footprint in as an array to create your kernel.

```
# Standard Imports
import numpy as np
from scipy.ndimage import convolve as sp_convolve
from scipy.ndimage import laplace

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

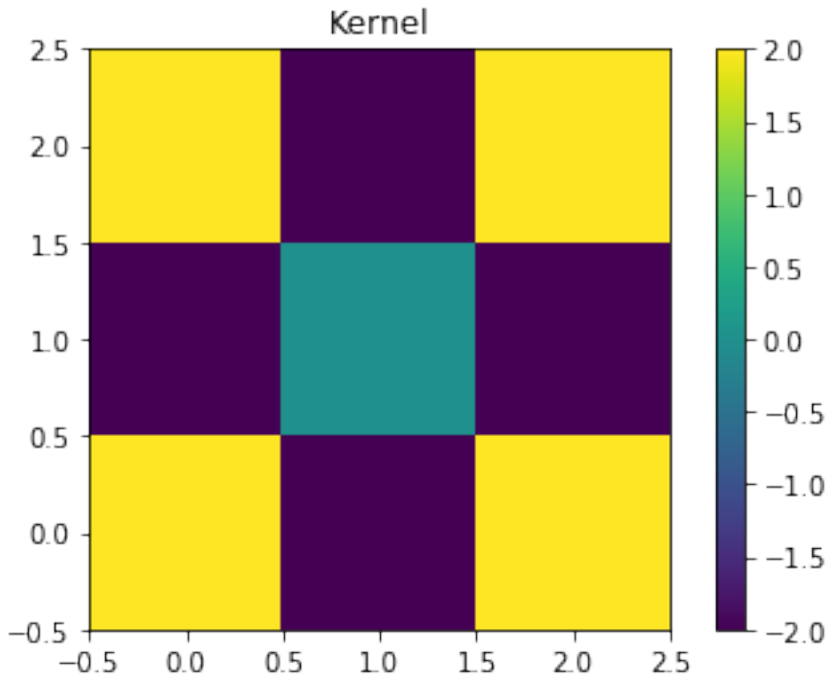
```
# grab subsection of fits images
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
sci1 = fits.getdata(test_data,ext=1)
my_arr = sci1[700:1030,2250:2800]

# setup our laplace kernel with a target footprint (diagonals in IRAF)
footprint = np.array([[0, 1, 0], [1, 1, 1], [0, 1, 0]])
laplace_kernel = laplace(footprint)
```

(continues on next page)

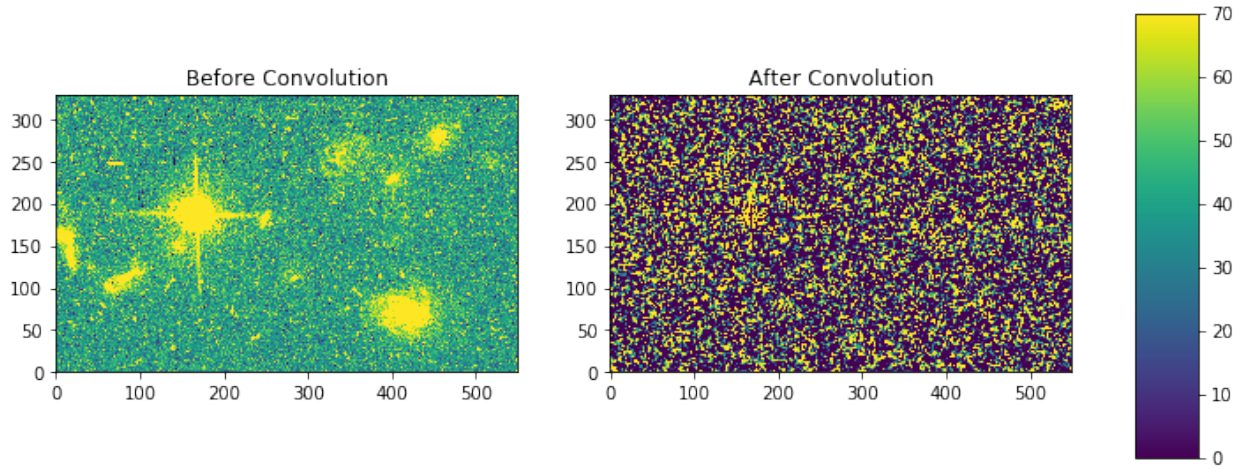
(continued from previous page)

```
# perform scipy convolution
result = sp_convolve(my_arr, laplace_kernel)
```

```
plt.imshow(laplace_kernel, interpolation='none', origin='lower')
plt.title('Kernel')
plt.colorbar()
plt.show()
```




```
fig, axes = plt.subplots(nrows=1, ncols=2)
a = axes[0].imshow(my_arr,interpolation='none', origin='lower',vmin=0, vmax=70)
axes[0].set_title('Before Convolution')
a = axes[1].imshow(result,interpolation='none', origin='lower',vmin=0, vmax=70)
axes[1].set_title('After Convolution')

fig.subplots_adjust(right = 0.8,left=0)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(a, cax=cbar_ax)
fig.set_size_inches(10,5)
plt.show()
```



2.1.6 median-rmedian

Please review the *Notes* section above before running any examples in this notebook

Apply a median filter to your data array, and save the smoothed image back out to a FITS file. We will use the `scipy.ndimage.filters.median_filter` function.

```
# Standard Imports
import numpy as np
from scipy.ndimage.filters import median_filter

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

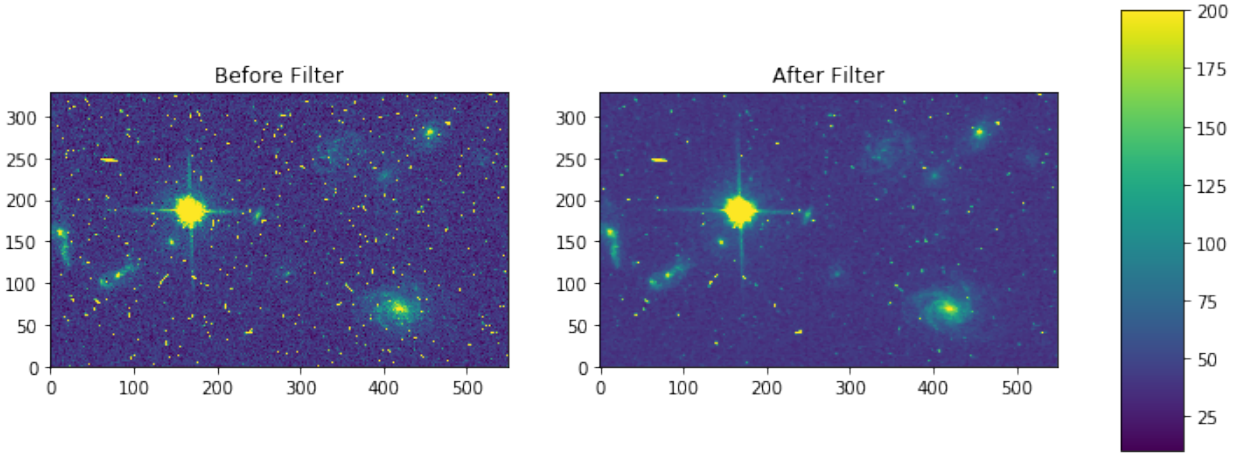
```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
# create test array
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
out_file = 'median_out.fits'
sci1 = fits.getdata(test_data,ext=1)
my_arr = sci1[700:1030,2250:2800]

# apply median filter
filtered = median_filter(my_arr,size=(3,4))
```

```
# save smoothed image to a new FITS file
hdu = fits.PrimaryHDU(filtered)
hdu.writeto(out_file, overwrite=True)
```

```
fig, axes = plt.subplots(nrows=1, ncols=2)
pmin,pmax = 10, 200
a = axes[0].imshow(my_arr,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[0].set_title('Before Filter')
a = axes[1].imshow(filtered,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[1].set_title('After Filter')

fig.subplots_adjust(right = 0.8,left=0)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(a, cax=cbar_ax)
fig.set_size_inches(10,5)
plt.show()
```



For a ring median filter we can supply a more specific footprint to the `median_filter` function. You can easily generate this footprint using the `astropy.convolution` library. In this example we will also show how to use the equivalent of the IRAF zloreject/zhireject parameter. The handling of `numpy nan` values is only available with the `Astropy` convolution.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astropy.convolution import convolve as ap_convolve
from astropy.convolution import Ring2DKernel

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

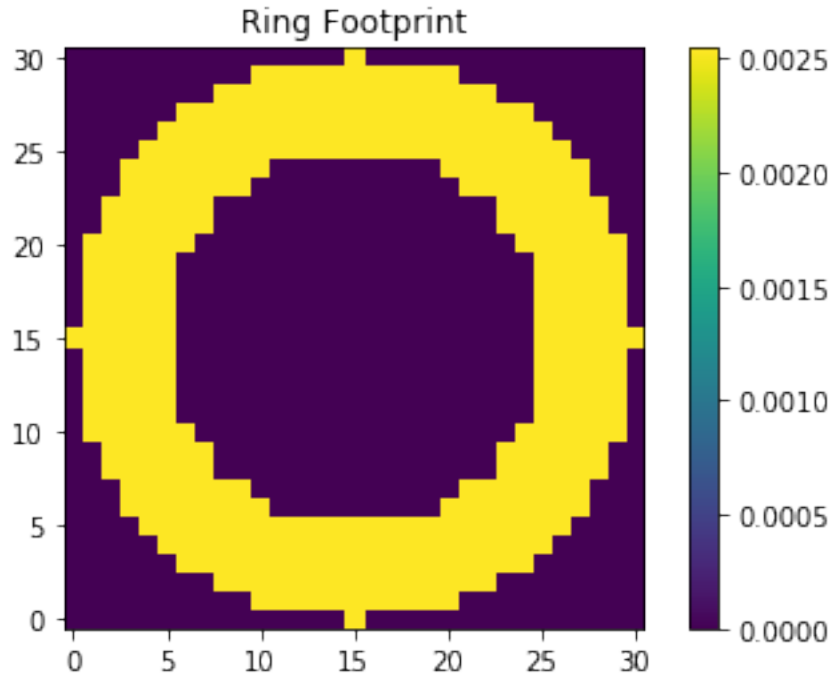
```
# create test array
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
sci1 = fits.getdata(test_data,ext=1)
my_arr = sci1[700:1030,2250:2800]
```

(continues on next page)

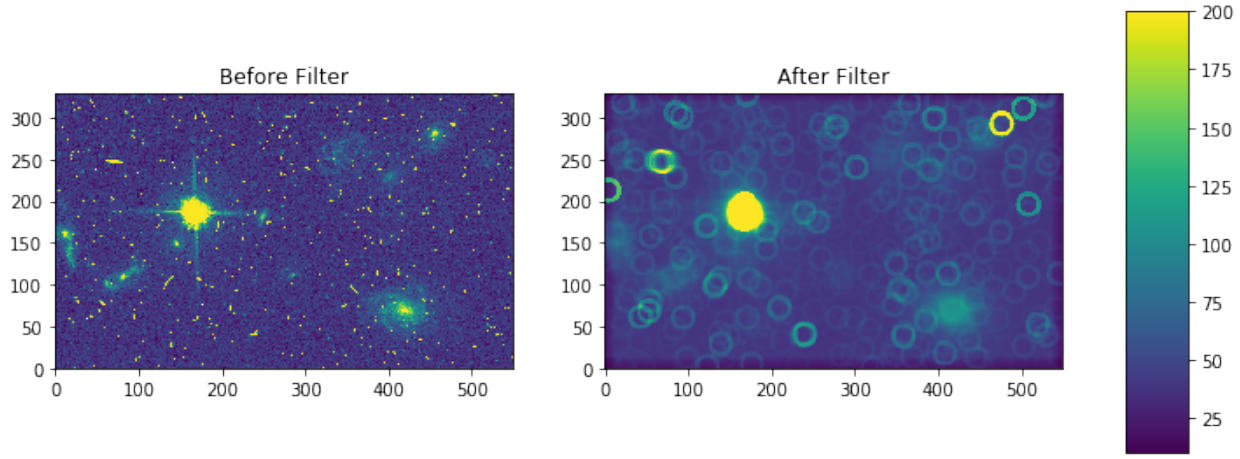
(continued from previous page)

```
# create ring filter
ringKernel = Ring2DKernel(10,5)

# apply a zloreject value by setting certain values to numpy nan
my_arr[my_arr < -99] = np.nan

# apply median filter
filtered = ap_convolve(my_arr, ringKernel, normalize_kernel=True)
```

```
plt.imshow(ringKernel, interpolation='none', origin='lower')
plt.title('Ring Footprint')
plt.colorbar()
plt.show()
```




```
fig, axes = plt.subplots(nrows=1, ncols=2)
pmin,pmax = 10, 200
a = axes[0].imshow(my_arr,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[0].set_title('Before Filter')
a = axes[1].imshow(filtered,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[1].set_title('After Filter')

fig.subplots_adjust(right = 0.8,left=0)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(a, cax=cbar_ax)
fig.set_size_inches(10,5)
plt.show()
```



2.1.7 mode-rmode

Please review the *Notes* section above before running any examples in this notebook

The mode calculation equation used in the mode and rmode IRAF tasks (3.0*median - 2.0*mean) can be recreated using the scipy.ndimage.generic_filter function. The equation was used as an approximation for a mode calculation.

```
# Standard Imports
import numpy as np
from scipy.ndimage import generic_filter

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
def mode_func(in_arr):
    f = 3.0*np.median(in_arr) - 2.0*np.mean(in_arr)
    return f
```

For a box footprint:

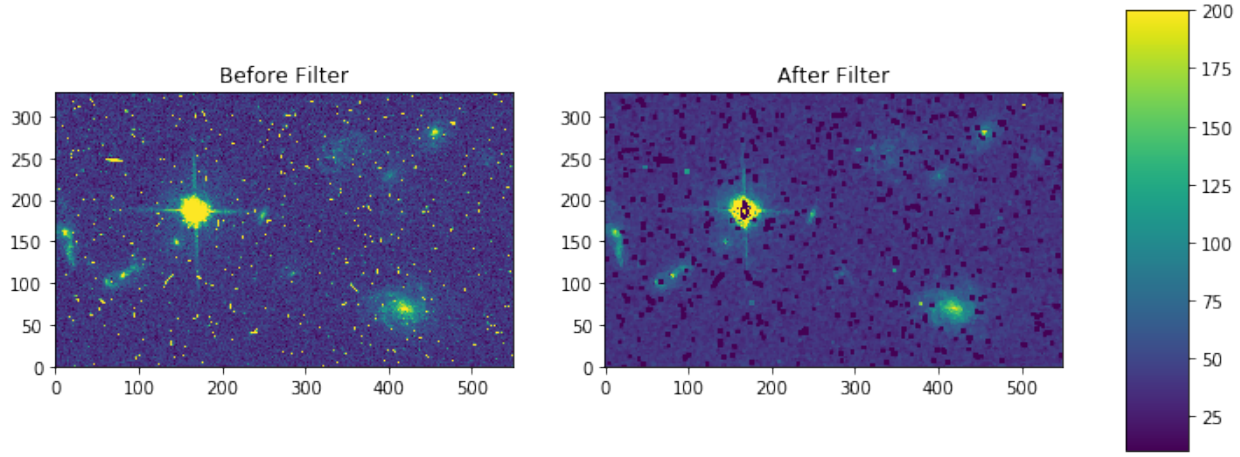
```
# create test array
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
sci1 = fits.getdata(test_data,ext=1)
my_arr = sci1[700:1030,2250:2800]
```

(continues on next page)

(continued from previous page)

```
# apply mode filter
filtered = generic_filter(my_arr, mode_func, size=5)
```

```
fig, axes = plt.subplots(nrows=1, ncols=2)
pmin,pmax = 10, 200
a = axes[0].imshow(my_arr,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[0].set_title('Before Filter')
a = axes[1].imshow(filtered,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[1].set_title('After Filter')

fig.subplots_adjust(right = 0.8,left=0)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(a, cax=cbar_ax)
fig.set_size_inches(10,5)
plt.show()
```



For a ring footprint (similar to IRAF's rmode):

```
# Standard Imports
import numpy as np
from scipy.ndimage import generic_filter

# Astronomy Specific Imports
from astropy.io import fits
from astroimtools import circular_annulus_footprint

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

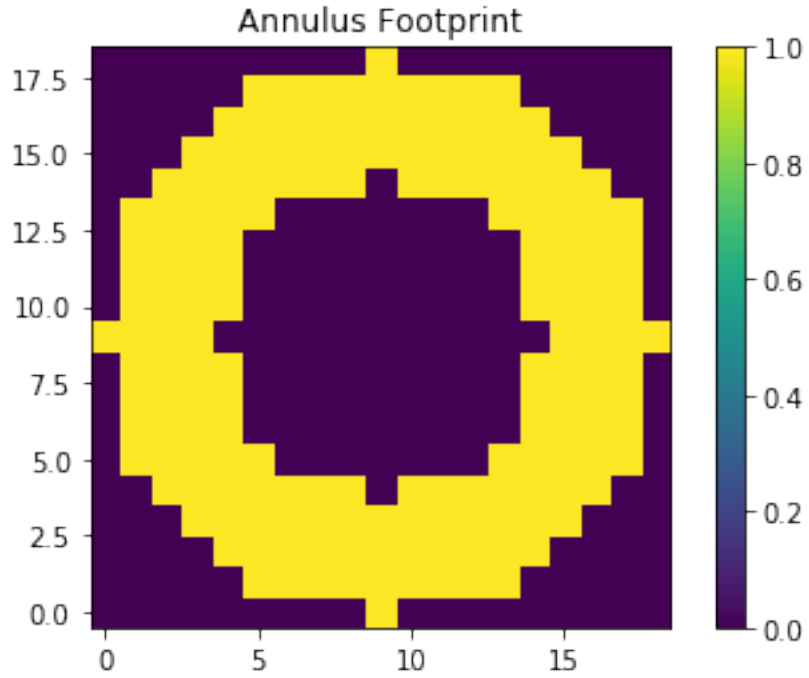
```
# create test array
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
sci1 = fits.getdata(test_data,ext=1)
my_arr = sci1[700:1030,2250:2800]

# create annulus filter
fp = circular_annulus_footprint(5, 9)
# apply mode filter
```

(continues on next page)

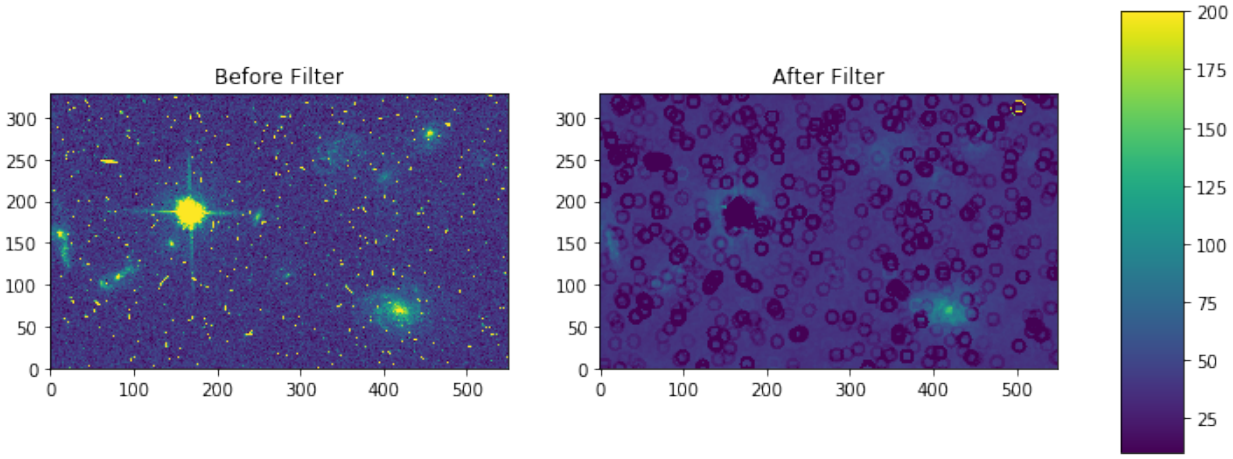
(continued from previous page)

```
filtered = generic_filter(my_arr, mode_func, footprint=fp)
```

```
plt.imshow(fp, interpolation='none', origin='lower')
plt.title('Annulus Footprint')
plt.colorbar()
plt.show()
```




```
fig, axes = plt.subplots(nrows=1, ncols=2)
pmin,pmax = 10, 200
a = axes[0].imshow(my_arr,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[0].set_title('Before Filter')
a = axes[1].imshow(filtered,interpolation='none', origin='lower',vmin=pmin, vmax=pmax)
axes[1].set_title('After Filter')

fig.subplots_adjust(right = 0.8,left=0)
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(a, cax=cbar_ax)
fig.set_size_inches(10,5)
plt.show()
```



2.1.8 Not Replacing

- runmed - see **images.imutil.imsum**
- fmode - see *images.imfilter.mode*
- fmedian - see *images.imfilter.median*
- gradient - **may** replace in future

2.2 images.imfit

images.imfit contains three functions for fitting and value replacement.

2.2.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

The fitting and replacement functionality of images.imfit can be replaced using tools available in Astropy and Scipy. Please see the linked doc pages for more details. These fitting functions have a lot more options and function types then what was available in IRAF.

All tasks in the images.imfit package provide the same function type options. Here is a conversion for these functions in Python:

- leg - astropy.modeling.polynomial.Legendre1D (or 2D)
- cheb - astropy.modeling.polynomial.Chebyshev1D (or 2D)
- spline1 - scipy.interpolate.UnivariantSpline
- spline3 - scipy.interpolate.CubicSpline

For ngrow the option (not covered here) see `disk` and `dilation` from skimage.morphology

Contents:

- *fit1d-lineclean*

- *imsurfit*

```
# Temporarily change default colormap to viridis
import matplotlib.pyplot as plt
plt.rcParams['image.cmap'] = 'viridis'
```

2.2.2 fit1d-lineclean

Please review the *Notes* section above before running any examples in this notebook

The fit1d task will fit a function to a 1d data array. The lineclean task will take this same fitting tasks, and return your data array with values outside a certain sigma limit replaced with the fitted values. We can preform both these tasks with `astropy` and `scipy` functions. You'll find many more models defined in the `astropy.models` library here

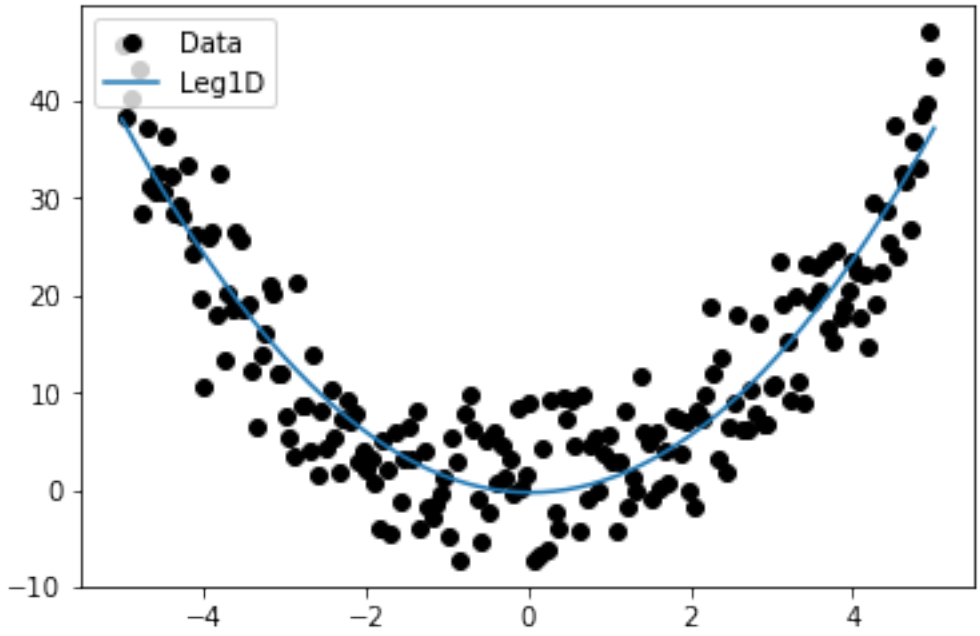
Below we show one example using a Legendre 1D fit, and another example using a linear spine. For the linear spline example we will go over an implementation of lineclean.

```
# Standard Imports
import numpy as np
from scipy.interpolate import UnivariateSpline

# Astronomy Specific Imports
from astropy.modeling import models, fitting, polynomial

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Fitting - Legendre1D
# This example is taken in part from examples on the Astropy Modeling documentation

# Generate fake data
np.random.seed(0)
x = np.linspace(-5., 5., 200)
y = 0.5 * (3*x**2-1)
y += np.random.normal(0., 5, x.shape)

# Fit the data using the Legendre1D fitter
leg_init = polynomial.Legendre1D(2)
fit_leg = fitting.LevMarLSQFitter()
leg = fit_leg(leg_init, x, y)

# Plot solution
plt.plot(x, y, 'ko', label='Data')
plt.plot(x, leg(x), label='Leg1D')
plt.legend(loc=2)
```

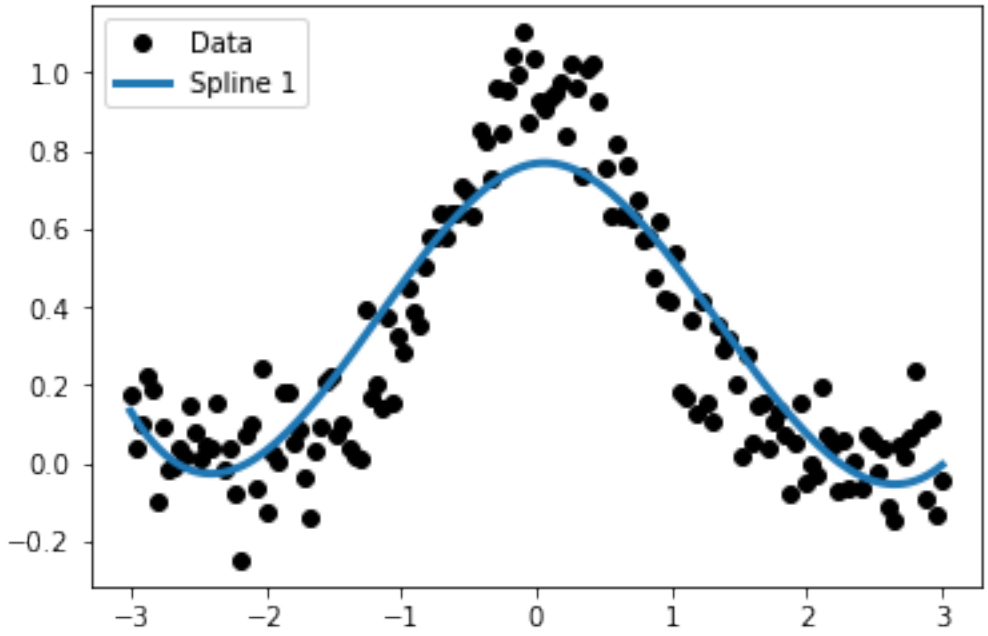
```
WARNING: Model is linear in parameters; consider using linear fitting methods.␣
↪[astropy.modeling.fitting]
```

```
<matplotlib.legend.Legend at 0x11a7a4908>
```




```
# Fitting - Spline1
# This example is taken in part from examples on the Astropy Modeling documentation

# Generate fake data
np.random.seed(0)
x = np.linspace(-3., 3., 150)
y = np.exp(-x**2) + 0.1 * np.random.randn(150)

# Fit the data using the Spline 1 fitter
spl1 = UnivariateSpline(x, y)
spl1.set_smoothing_factor(3)

# Plot solution
plt.plot(x, y, 'ko', label='Data')
plt.plot(x, spl1(x),lw=3,label='Spline 1')
plt.legend(loc=2)
```

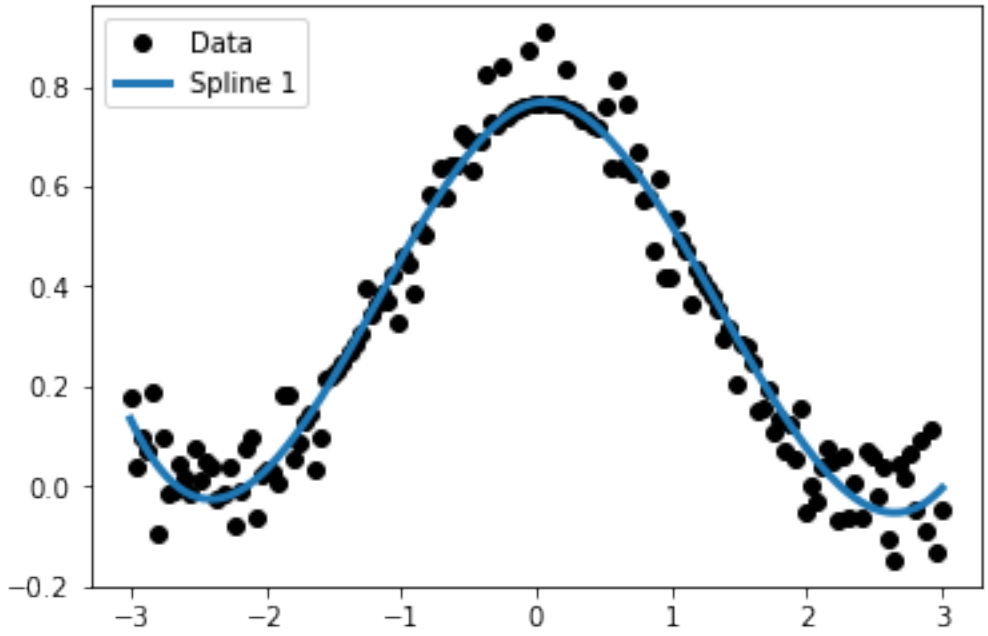
```
<matplotlib.legend.Legend at 0x11a889b38>
```




```
# Fitting and replacement of outlier values - Spline 1

# Fit array
fit_data = spl1(x)
residuals = fit_data-y
sigma = np.std(residuals)

# Let's reject everything outside of 1-sigma of the fit residuals
boolean_array = [np.absolute(residuals) > sigma]
y[boolean_array] = fit_data[boolean_array]
```

```
# Plot solution
plt.plot(x, y, 'ko', label='Data')
plt.plot(x, spl1(x),lw=3,label='Spline 1')
plt.legend(loc=2)
```

```
<matplotlib.legend.Legend at 0x11a9950f0>
```



2.2.3 imsurfit

Please review the *Notes* section above before running any examples in this notebook

Imsurfit has similiar functionality to the above tasks, but in 2 dimensions. Below we show a brief example, which can be extended as shown above in the *lineclean* example. We use the `Polynomial2D` `astropy.modeling` example here to showcase the usage of models not found in this IRAF library.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.modeling import models, fitting

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Fitting - Polynomial2D
# This example is taken from the Astropy Modeling documentation

# Generate fake data
np.random.seed(0)
y, x = np.mgrid[:128, :128]
z = 2. * x ** 2 - 0.5 * y ** 2 + 1.5 * x * y - 1.
z += np.random.normal(0., 0.1, z.shape) * 50000.

# Fit the data using astropy.modeling
p_init = models.Polynomial2D(degree=2)
fit_p = fitting.LevMarLSQFitter()
p = fit_p(p_init, x, y, z)
```

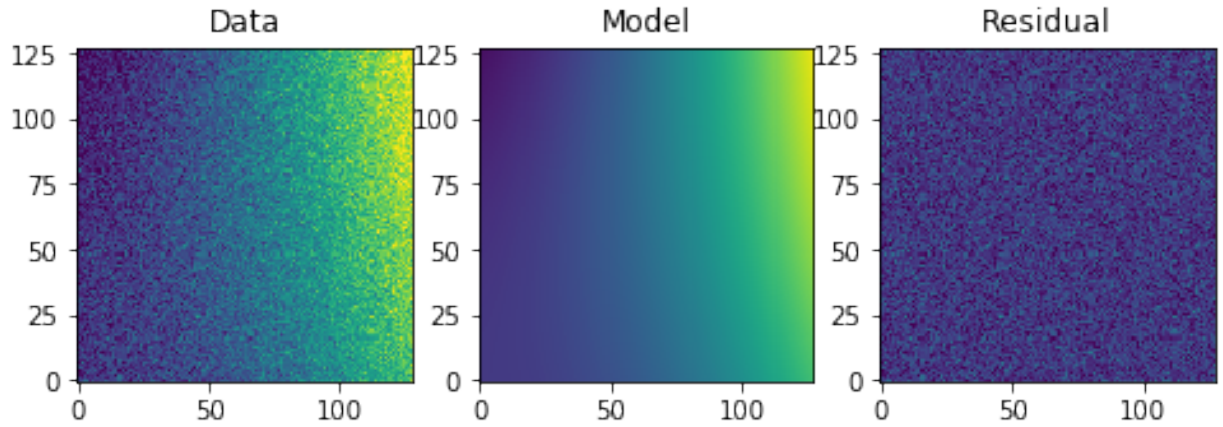
(continues on next page)

(continued from previous page)

```
# Plot the data with the best-fit model
plt.figure(figsize=(8, 2.5))
plt.subplot(1, 3, 1)
plt.imshow(z, origin='lower', interpolation='nearest', vmin=-1e4, vmax=5e4)
plt.title("Data")
plt.subplot(1, 3, 2)
plt.imshow(p(x, y), origin='lower', interpolation='nearest', vmin=-1e4,
           vmax=5e4)
plt.title("Model")
plt.subplot(1, 3, 3)
plt.imshow(z - p(x, y), origin='lower', interpolation='nearest', vmin=-1e4,
           vmax=5e4)
plt.title("Residual")
```

```
WARNING: Model is linear in parameters; consider using linear fitting methods.␣
↪[astropy.modeling.fitting]
```

```
<matplotlib.text.Text at 0x119e94ac8>
```



2.3 images.imgeom

The images.imgeom package contains various image spatial manipulation and interpolation tasks.

2.3.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

General Note about this package:

The tasks in this IRAF package include support for WCS updates after the image manipulations are preformed. Efforst in the community are ongoing for these WCS capabilities. For the moment we have included the array manipulation part of these tasks in this notebook.

Boundary Condition and Interpolation Options:

In the IRAF package the interpolation options are as follows: nearest, linear, poly3, poly5, spline3, sinc/lsinc. In the `scipy.ndimage` functions, the interpolation option are spline degrees 0-5, where spline-0 is nearest and spline-1 is linear.

The boundary condition options for IRAF and `scipy` are the same: nearest, wrap, reflect, and constant.

Important Note to Users: There are some differences in algorithms between some of the IRAF and Python Interpolations. Proceed with care if you are comparing prior IRAF results to Python results. For more details on this issue see the filed Github issue.

Contents:

2.3.2 blkavg

Please review the *Notes* section above before running any examples in this notebook

The blkavg task takes in an arbitrary dimensioned image and performs a block average across the requested box size. We can preform the same task with the `astropy.nddata.utils.block_reduce` function. In fact, this function is more generalized as you can apply any function (not just averaging) that accepts an `ndarray` and axis keyword to the block filter.

Below we show an example that reads in a FITS file, runs `block_reduce` on the first image extension, and saves out the updated file.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astropy.nddata.utils import block_reduce
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
# Change this value to your desired data file, here were creating a filename
# for our new changed data
orig_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
new_fits = 'iczgs3ygq_newdtype_flt.fits'

# Read in your fits file
```

(continues on next page)

(continued from previous page)

```
hdu = fits.open(orig_data)

# Print FITS summary
hdu.info()

# Run block reduce on first sci extension
red_data = block_reduce(hdu[1].data, [2,2], func=np.mean)

# Re-insert the data into the HDUList
hdu[1].data = red_data

# Save out fits to a new file
hdu.writeto(new_fits, overwrite=True)
hdu.close()
```

```
Filename: ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     279   ()
  1  SCI           1 ImageHDU       200   (4096, 2048)   float32
  2  ERR           1 ImageHDU        56   (4096, 2048)   float32
  3  DQ            1 ImageHDU        48   (4096, 2048)   int16
  4  SCI           2 ImageHDU       198   (4096, 2048)   float32
  5  ERR           2 ImageHDU        56   (4096, 2048)   float32
  6  DQ            2 ImageHDU        48   (4096, 2048)   int16
  7  D2IMARR       1 ImageHDU        15   (64, 32)   float32
  8  D2IMARR       2 ImageHDU        15   (64, 32)   float32
  9  D2IMARR       3 ImageHDU        15   (64, 32)   float32
 10  D2IMARR       4 ImageHDU        15   (64, 32)   float32
 11  WCSDVARR      1 ImageHDU        15   (64, 32)   float32
 12  WCSDVARR      2 ImageHDU        15   (64, 32)   float32
 13  WCSDVARR      3 ImageHDU        15   (64, 32)   float32
 14  WCSDVARR      4 ImageHDU        15   (64, 32)   float32
 15  WCSCORR       1 BinTableHDU     59   14R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
```

2.3.3 blkrep

Please review the *Notes* section above before running any examples in this notebook

The task `blkrep` is used to block replicate an n-dimensional image. Astropy has the equivalent function `block_replicate`.

For an example of how to read in a FITS extension, edit the image array, and save out the updated file, see *blkavg* above.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.nddata.utils import block_replicate
```

```
# test data
my_arr = np.array(([[0., 1.], [2., 3.]]))
print("input array")
print(my_arr)
```

(continues on next page)

(continued from previous page)

```
# conservation of the array sum is the default
out = block_replicate(my_arr, 3)
print("sum conservation")
print(out)

# you can changes this using conserve_sum=False
out = block_replicate(my_arr, 3, conserve_sum=False)
print("no sum conservation")
print(out)
```

```
input array
[[ 0.  1.]
 [ 2.  3.]]
sum conservation
[[ 0.          0.          0.          0.11111111  0.11111111  0.11111111]
 [ 0.          0.          0.          0.11111111  0.11111111  0.11111111]
 [ 0.          0.          0.          0.11111111  0.11111111  0.11111111]
 [ 0.22222222  0.22222222  0.22222222  0.33333333  0.33333333  0.33333333]
 [ 0.22222222  0.22222222  0.22222222  0.33333333  0.33333333  0.33333333]
 [ 0.22222222  0.22222222  0.22222222  0.33333333  0.33333333  0.33333333]]
no sum conservation
[[ 0.  0.  0.  1.  1.  1.]
 [ 0.  0.  0.  1.  1.  1.]
 [ 0.  0.  0.  1.  1.  1.]
 [ 2.  2.  2.  3.  3.  3.]
 [ 2.  2.  2.  3.  3.  3.]
 [ 2.  2.  2.  3.  3.  3.]]
```

2.3.4 im3dtran-imtranspose

Please review the *Notes* section above before running any examples in this notebook

Tasks used to transpose images. numpy.transpose can handle any number of dimensions.

For an example of how to read in a FITS extension, edit the image array, and save out the updated file, see *blkavg* above.

```
# Standard Imports
import numpy as np
```

```
# in_array constructs a 3 x 5 array of integer values from 0 to 14
in_array = np.arange(15).reshape(5,3)
# we then transpose it it to a 5 x 3 array
out_array = np.transpose(in_array)

print('Original array:')
print(in_array)
print('Transpose of original array')
print(out_array)
```

```
Original array:
[[ 0  1  2]
 [ 3  4  5]
```

(continues on next page)

(continued from previous page)

```
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]
Transpose of original array
[[ 0  3  6  9 12]
 [ 1  4  7 10 13]
 [ 2  5  8 11 14]]
```

2.3.5 imshift-shiftlines

Please review the *Notes* section above before running any examples in this notebook

The task imshift can shift an image in x and y by float values and will use interpolation to create the output image. Shiftlines preformed similar functionality but We will be using scipy.ndimage.shift, where you can shift in any axis of your image. See the *Notes* at the top of the notebook for fitting and boundary options.

For an example of how to read in a FITS extension, edit the image array, and save out the updated file, see *blkavg* above.

```
# Standard Imports
import numpy as np
from scipy.ndimage import shift
```

```
# Don't forget that Python uses (y,x) format when specifiying shifts
in_array = np.arange(25).reshape(5,5)
out_array = shift(x, (0.8,0.8), order=3, mode='constant', cval=2)

print('Original array:')
print(in_array)
print('A zoom of 0.5 in y and 2 in x with nearest')
print(out_array)
```

```
Original array:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
A zoom of 0.5 in y and 2 in x with nearest
[[ 2  2  2  2  2]
 [ 2  0  2  2  4]
 [ 2  6  7  8  9]
 [ 2 11 12 13 14]
 [ 2 16 18 19 20]]
```

2.3.6 magnify

Please review the *Notes* section above before running any examples in this notebook

The task magnify takes an image and magnifies the image by the desired amount, using a chosen iterpolation. The interpolation options avaialable for the magnify task are nearest, linear, poly3, poly5, spine3, sinc, lsinc, and drizzle. We will be using scipy.ndimage.zoom as a python equivalent. For this task, the available interpolation options are nearest, and spline0-5 fits.

For an example of how to read in a FITS extension, edit the image array, and save out the updated file, see *blkavg* above.

```
# Standard Imports
import numpy as np
from scipy.ndimage import zoom
```

```
# Don't forget that Python uses (y,x) format when specifiying magnification
in_array = np.arange(25).reshape(5,5)
out_array = zoom(in_array, (0.5,2.5), order=0)

print('Original array:')
print(in_array)
print('A zoom of 0.5 in y and 2.5 in x with nearest')
print(out_array)
```

```
Original array:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
A zoom of 0.5 in y and 2.5 in x with nearest
[[ 0  0  1  1  1  2  2  3  3  3  4  4]
 [20 20 21 21 21 22 22 23 23 23 24 24]]
```

2.3.7 rotate

Please review the *Notes* section above before running any examples in this notebook

The task rotate is used to rotate and shift images. We will only cover rotation here, for shifting please see *shiftlines*. We will be using scipy.ndimage.rotate for rotation using interpolation. For a simple 90 degree unit rotation we will use numpy.rot90 (you can do a 90 degree rotation using `scipy.ndimage.rotate`).

For an example of how to read in a FITS extension, edit the image array, and save out the updated file, see *blkavg* above.

Rotation using interpolation:

```
# Standard Imports
import numpy as np
from scipy.ndimage import rotate
```

```
in_array = np.arange(25).reshape(5,5)
# Rotate by 60 degrees
out_array = rotate(in_array, 60, axes=(1,0))
```

(continues on next page)

(continued from previous page)

```
print('Original array:')
print(in_array)
print('A rotation of 60 degrees')
print(out_array)
```

```
Original array:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
A rotation of 60 degrees
[[ 0  0  0  0  0  0  0]
 [ 0  0  3  9  0  0  0]
 [ 0  0  5 11 15 21  0]
 [ 0  2  7 12 17 22  0]
 [ 0  3  9 13 19  0  0]
 [ 0  0  0 15 21  0  0]
 [ 0  0  0  0  0  0  0]]
```

Rotation in increments of 90 degrees:

```
# Standard Imports
import numpy as np
```

```
in_array = np.arange(25).reshape(5,5)
# Rotate by 270 degrees
out_array = np.rot90(in_array, 3)

print('Original array:')
print(in_array)
print('A rotation of 270 degrees')
print(out_array)
```

```
Original array:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
A rotation of 270 degrees
[[20 15 10  5  0]
 [21 16 11  6  1]
 [22 17 12  7  2]
 [23 18 13  8  3]
 [24 19 14  9  4]]
```

2.3.8 Not Replacing

- imlintran - see *images.imgeom.magnify*, *images.imgeom.rotate*, and *images.imgeom.imshift*

2.4 images.imutil

The images.imutil package provides general FITS image tools such as header editing and image arithmetic.

2.4.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Contents:

2.4.2 chpixtype

Please review the *Notes* section above before running any examples in this notebook

Chpixtype is a task that allows you to change the pixel type of a FITS image. There is built in functionality in `astropy.io.fits` to preform this task with the `scale` method. Below you will find a table that translates the chpixtype newpixtype options into their equivalent numpy/astropy type.

Type Conversions

Chpixtype	Numpy/Astropy Type
ushort	uint16
short	int16
int	int32
long	int64
real	float32
double	float64

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
Downloading URL https://mast.stsci.edu/api/v0/download/file?uri=mast:HST/product/
↪iczgs3ygq/iczgs3ygq_flt.fits to ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits ...␣
↪[Done]
```

```
# Change this value to your desired data file, here were creating a filename
# for our new changed data
orig_data = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
new_data = 'iczgs3ygq_newdtype_flt.fits'

# Read in your FITS file
hdu = fits.open(orig_data)

# Print info about FITS file
hdu.info()

# Edit the datatype for the first sci extension
hdu[1].scale(type='int32')

# Save changed hdu object to new file
# The overwrite argument tells the writeto method to overwrite if file already exists
hdu.writeto(new_data, overwrite=True)
hdu.close()
```

```
Filename: ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     265   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
  3  DQ            1 ImageHDU        43   (1014, 1014)   int16
  4  SAMP          1 ImageHDU        37   (1014, 1014)   int16
  5  TIME          1 ImageHDU        37   (1014, 1014)   float32
  6  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
```

2.4.3 hedit

Please review the *Notes* section above before running any examples in this notebook

The hedit task allows users to edit an image header. This functioanlity is covered in `astropy.io.fits`. Take note that to make changes to a FITS file, you must use the `mode='update'` keyword in the `fits.open` call. The default mode for `fits.open` is `readonly`. Below you'll find examples of editing a keyword if it does/doesn't exist, and how to delete keywords from the header. Also provided is an example of updating multiple files at once using the convience function setval.

For examples on printing/viewing header keywords please see *hselect*

```
# Standard Imports
from glob import glob

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Change this value to your desired data file
test_data = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'

# Open FITS file, include the mode='update' keyword
hdu = fits.open(test_data, mode='update')

# Simple header change, will add keyword if it doesn't exist
hdu[0].header['MYKEY1'] = 'Editing this keyword'

# Only add keyword if it does not already exist:
if 'MYKEY2' not in hdu[0].header:
    hdu[0].header['MYKEY2'] = 'Also editing this'

# To delete keywords, first check if they exist:
if 'MYKEY2' in hdu[0].header:
    del hdu[0].header['MYKEY2']

# Close FITS file, this will save your changes
hdu.close()
```

Below we will show an example of how to update a keyword in multiple FITS files using the Astropy convenience function astropy.io.fits.setval and the glob function. `Astropy.io.fits.setval` will add the keyword if it does not already exist.

```
# Change this value to your desired search
data_list = glob('./mastDownload/HST/ICZGS3YGQ/*.fits')

# Now we loop over the list of file and use the setval function to update keywords
# Here we update the keyword MYKEY1 value to the integer 5.
```

(continues on next page)

(continued from previous page)

```
for filename in data_list:
    fits.setval(filename, 'MYKEY1', value=5)
```

2.4.4 hselect

Please review the *Notes* section above before running any examples in this notebook

The hselect task allows users to search for keyword values in the FITS headers. This functionality has been replaced by the CCDProc ImageFileCollection class. This class stores the header keyword values in an Astropy Table object. There is also an executable script provided by Astropy called fitsheader. You'll find examples of both below.

If you wish to save your output to a text file, please see the Astropy Table Documentation and the Astropy Unified I/O page.

```
# Astronomy Specific Imports
from ccdproc import ImageFileCollection
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid, productFilename="jczgx1ppq_flc.fits")
obsid = '2004663554'
Observations.download_products(obsid, productFilename="jczgx1ptq_flc.fits")
obsid = '2004663556'
Observations.download_products(obsid, productFilename="jczgx1q1q_flc.fits")

import shutil
shutil.move('./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits','../data/')
shutil.move('./mastDownload/HST/JCZGX1PTQ/jczgx1ptq_flc.fits','../data/')
shutil.move('./mastDownload/HST/JCZGX1Q1Q/jczgx1q1q_flc.fits','../data/')
```

```
INFO:astropy:Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with␣
↪expected size 167964480.
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
INFO:astropy:Found cached file ./mastDownload/HST/JCZGX1PTQ/jczgx1ptq_flc.fits with␣
↪expected size 167964480.
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PTQ/jczgx1ptq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
INFO:astropy:Found cached file ./mastDownload/HST/JCZGX1Q1Q/jczgx1q1q_flc.fits with␣
↪expected size 167964480.
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1Q1Q/jczgx1q1q_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
'../data/jczgx1q1q_flc.fits'
```

```
# first we make the ImageFileCollection object
collec = ImageFileCollection('../data/',
                             keywords=["filetype","date","exptime","filter2"],
                             glob_include="jcz*.fits", ext=0)

# header keywords values are stored in an Astropy Table in the summary attribute
out_table = collec.summary
out_table
```

```
# Now we can filter our table based on keyword values using Python bitwise operators
filtered_table = out_table[(out_table['exptime'] < 600) & (out_table['filter2'] ==
↪'F814W')]
filtered_table
```

```
# Now let's extract the filename list from our filtered table into a python List␣
↪object
filelist = filtered_table['file'].data
print(filelist)

for filename in filelist:
    print(filename)
    # Do your analysis here
```

```
['jczgx1ppq_flc.fits' 'jczgx1q1q_flc.fits']
jczgx1ppq_flc.fits
jczgx1q1q_flc.fits
```

Also available is the Astropy executable script fitsheader. Fitsheader can be run from the command line.

```
# the "!" character tells the notebook to run this command as if it were in a␣
↪terminal window
!fitsheader --help
```

```
usage: fitsheader [-h] [-e HDU] [-k KEYWORD] [-t [FORMAT]] [-c]
                  filename [filename ...]

Print the header(s) of a FITS file. Optional arguments allow the desired
extension(s), keyword(s), and output format to be specified. Note that in the
case of a compressed image, the decompressed header is shown by default.

positional arguments:
  filename              path to one or more files; wildcards are supported

optional arguments:
  -h, --help            show this help message and exit
  -e HDU, --extension HDU
                        specify the extension by name or number; this argument
                        can be repeated to select multiple extensions
  -k KEYWORD, --keyword KEYWORD
                        specify a keyword; this argument can be repeated to
                        select multiple keywords; also supports wildcards
  -t [FORMAT], --table [FORMAT]
                        print the header(s) in machine-readable table format;
                        the default format is "ascii.fixed_width" (can be
                        "ascii.csv", "ascii.html", "ascii.latex", "fits", etc)
  -c, --compressed      for compressed image data, show the true header which
```

(continues on next page)

(continued from previous page)

```
                    describes the compression rather than the data
```

```
# print out only the keyword names that match FILE* or NAXIS*
!fitsheader --keyword FILE* --keyword NAXIS* ../data/*.fits
```

```
# HDU 0 in ../data/imstack_out.fits:
NAXIS   =                    3 / number of array dimensions
NAXIS1  =                 4096
NAXIS2  =                 2048
NAXIS3  =                    2
# HDU 0 in ../data/jczgx1ppq_flc.fits:
FILENAME= 'jczgx1ppq_flc.fits' / name of file
FILETYPE= 'SCI      '          / type of data found in data file
NAXIS   =                    0 / number of data axes

# HDU 1 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 2 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 3 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 4 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 5 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 6 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 7 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 8 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 9 in ../data/jczgx1ppq_flc.fits:
```

(continues on next page)

(continued from previous page)

```
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 10 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 11 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 12 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 13 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 14 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 15 in ../data/jczgx1ppq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                  455 / length of dimension 1
NAXIS2  =                   14 / length of dimension 2
# HDU 0 in ../data/jczgx1ptq_flc.fits:
FILENAME= 'jczgx1ptq_flc.fits' / name of file
FILETYPE= 'SCI      '          / type of data found in data file
NAXIS   =                    0 / number of data axes

# HDU 1 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 2 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 3 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 4 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2
```

(continues on next page)

(continued from previous page)

```
# HDU 5 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 6 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 7 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 8 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 9 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 10 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 11 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 12 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 13 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 14 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 15 in ../data/jczgx1ptq_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                  455 / length of dimension 1
NAXIS2  =                   14 / length of dimension 2
# HDU 0 in ../data/jczgx1q1q_flc.fits:
FILENAME= 'jczgx1q1q_flc.fits' / name of file
```

(continues on next page)

(continued from previous page)

```
FILETYPE= 'SCI      '           / type of data found in data file
NAXIS   =                    0 / number of data axes

# HDU 1 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 2 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 3 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 4 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 5 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 6 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2

# HDU 7 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 8 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 9 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 10 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 11 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32
```

(continues on next page)

(continued from previous page)

```
# HDU 12 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 13 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 14 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                   64
NAXIS2  =                   32

# HDU 15 in ../data/jczgx1q1q_flc.fits:
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                  455 / length of dimension 1
NAXIS2  =                   14 / length of dimension 2
```

```
# print out only the first extension and keyword names that match FILE* or NAXIS*
!fitsheader --extension 0 --keyword FILE* --keyword NAXIS* ../data/*.fits
```

```
# HDU 0 in ../data/imstack_out.fits:
NAXIS   =                    3 / number of array dimensions
NAXIS1  =                 4096
NAXIS2  =                 2048
NAXIS3  =                    2
# HDU 0 in ../data/jczgx1ppq_flc.fits:
FILENAME= 'jczgx1ppq_flc.fits' / name of file
FILETYPE= 'SCI      '          / type of data found in data file
NAXIS   =                    0 / number of data axes
# HDU 0 in ../data/jczgx1ptq_flc.fits:
FILENAME= 'jczgx1ptq_flc.fits' / name of file
FILETYPE= 'SCI      '          / type of data found in data file
NAXIS   =                    0 / number of data axes
# HDU 0 in ../data/jczgx1q1q_flc.fits:
FILENAME= 'jczgx1q1q_flc.fits' / name of file
FILETYPE= 'SCI      '          / type of data found in data file
NAXIS   =                    0 / number of data axes
```

2.4.5 imarith-imdivide

Please review the *Notes* section above before running any examples in this notebook

Imarith and imdivide both provide functionality to apply basic operators to whole image arrays. This task can be achieved with basic `astropy.io.fits` functionality along with `numpy` array functionality. We show a few examples below. In the first code cell we adding and dividing two image arrays together. In the second code cell we show how to use a data quality array to decide which image array values to replace with zero.

The basic operands (+,−,/,⋆) can all be used with an assignment operator in python (+=,−=,/=,⋆=). See http://www.tutorialspoint.com/python/python_basic_operators.htm for more details

```
# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615003'
Observations.download_products(obsid,productFilename="iczgs3y5q_flt.fits")
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Basic operands (+,-,/,*)
# Change these values to your desired data files
test_data1 = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
test_data2 = './mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits'
output_data = 'imarith_out.fits'
output_data2 = 'imarith_new.fits'


# Open FITS file
hdu1 = fits.open(test_data1)
hdu2 = fits.open(test_data2)

# Print information about the FITS file we opened
hdu1.info()
hdu2.info()

# Here we add hdu2-ext1 to hdu1-ext1 by using the shortcut += operator
hdu1[1].data += hdu2[1].data

# If you are dividing and need to avoid zeros in the image use indexing
indx_zeros = hdu2[1].data == 0
indx_nonzeros = hdu2[1].data != 0

# Set this value as you would the divzero parameter in imarith
# Here we're working with the error arrays of the image
set_zeros = 999.9
hdu1[2].data[indx_nonzeros] /= hdu2[2].data[indx_nonzeros]
hdu1[2].data[indx_zeros] = 999.9

# Save your new file
# The overwrite argument tells the writeto method to overwrite if file already exists
```

(continues on next page)

(continued from previous page)

```
hdu1.writeto(output_data, overwrite=True)

# If you want to save you updated array to a new file with just the updated image␣
↪array
# we can repackage the extension into a new HDUList
image_array = hdu1[1].data
new_hdu = fits.PrimaryHDU(image_array)
new_hdu.writeto(output_data2, overwrite=True)

# Close hdu files
hdu1.close()
hdu2.close()
```

```
Filename: ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     266   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
  3  DQ            1 ImageHDU        43   (1014, 1014)   int16
  4  SAMP          1 ImageHDU        37   (1014, 1014)   int16
  5  TIME          1 ImageHDU        37   (1014, 1014)   float32
  6  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
Filename: ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     265   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
  3  DQ            1 ImageHDU        43   (1014, 1014)   int16
  4  SAMP          1 ImageHDU        37   (1014, 1014)   int16
  5  TIME          1 ImageHDU        37   (1014, 1014)   float32
  6  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
```

```
# Here we show an example of using an HST DQ array to
# replace only certain values with zero in an image array

# Change these values to your desired data files
test_data1 = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
output_file = 'iczgs3ygq_updated.fits'

# Open FITS file
hdulist = fits.open(test_data1)

# First we should use the DQ array to make a boolean mask
DQ_mask = hdulist[3].data > 16384

# Now we can use the mask to replace values in the image array
# with 0.
hdulist[1].data[DQ_mask] = 0

# Now we can save out the edited FITS to a new file
hdulist.writeto(output_file)

# And finally, close the original FITS file
# The orignially file will not be updated since we did not
```

(continues on next page)

(continued from previous page)

```
# open the file in 'update' mode
hdulist.close()
```

2.4.6 imcopy

Please review the *Notes* section above before running any examples in this notebook

Imcopy allows users to copy a FITS image to a new file. We can accomplish this using `astropy.io.fits` by saving our FITS file to a new filename.

Imcopy will also make a cutout of an image and save the cutout to a new file with an updated WCS. We show an exampe of this in Python using the Cutout2D tool in `Astropy`. For more information on how to use `Cutout2D` please see this tutorial page.

```
# Astronomy Specific Imports
from astropy import wcs
from astropy.io import fits
from astropy.nddata import Cutout2D
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
obsid = '2004345211'
Observations.download_products(obsid,productFilename="jcw505010_drz.fits")
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
INFO:astropy:Found cached file ./mastDownload/HST/JCW505010/jcw505010_drz.fits with␣
↪expected size 219404160.
```

```
INFO: Found cached file ./mastDownload/HST/JCW505010/jcw505010_drz.fits with expected␣
↪size 219404160. [astroquery.query]
```

Simple example of a file copy

```
# Change these values to your desired filenames
test_data = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
output_data = 'imcopy_out.fits'

hdulist = fits.open(test_data)
# The overwrite argument tells the writeto method to overwrite if file already exists
hdulist.writeto(output_data, overwrite=True)
hdulist.close()
```

Example using a new cutout, here we will take a 50x50 pixel cutout from all image extensions centered at x:200, y:300

```
# Change these values to your desired filenames
test_data = './mastDownload/HST/JCW505010/jcw505010_drz.fits'
output_data = 'imcopy_cutout_out.fits'

hdulist = fits.open(test_data)

# Create iterable list of tuples to feed into Cutout2D,
# seperate list for extensions with wcs, as feeding the wcs
# back into the FITS file takes more work.
ext_list = [1,2]
for ext in ext_list:
    orig_wcs = wcs.WCS(hdulist[ext].header)
    cutout = Cutout2D(hdulist[ext].data, (200,300), (50,50), wcs=orig_wcs)
    hdulist[ext].data = cutout.data
    hdulist[ext].header.update(cutout.wcs.to_header())

hdulist.writeto(output_data, overwrite=True)

hdulist.close()
```

2.4.7 imfunction-imexpr

Please review the *Notes* section above before running any examples in this notebook

Imfunction will apply a function to the image pixel values in an image array. Imexpr gives you similiar functionality with the added capability to combine different images using a user created expression. We can accomplish this using the built in funcitonality of the numpy library.

If there is a particular function you would like to apply to your image array that you cannot find in the `numpy` library you can use the `np.vectorize` function, which can make any python function apply to each element of your array. But keep in mind that np.vectorize is esentially looping over the array, and may not be the most efficient method.

Example using exsisting numpy function:

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Change these values to your desired data files
test_data = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
```

(continues on next page)

(continued from previous page)

```
output_data = 'imfunction_out.fits'

# Here we use the cosine function as an example
hdu = fits.open(test_data)
sci = hdu[1].data

# When you call your new function, make sure to reassign the array to
# the new values if the original function is not changing values in place
hdu[1].data = np.cos(hdu[1].data)

# Now save out to a new file, and close the original file, changes will
# not be applied to the oiginal FITS file.
hdu.writeto(output_data, overwrite=True)
hdu.close()
```

Example using user defined function and `np.vectorize`:

```
# Change these values to your desired data files
test_data = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
output_data = 'imfunction2_out.fits'

# Here we use the following custom function as an example
def my_func(x):
    return (x**2)+(x**3)

# Now we open our file, and vectorize our function
hdu = fits.open(test_data)
sci = hdu[1].data
vector_func = np.vectorize(my_func)

# When you call your new function, make sure to reassign the array to
# the new values if the original function is not changing values in place
hdu[1].data = vector_func(hdu[1].data)

# Now save out to a new file, and close the original file, changes will
# not be applied to the oiginal FITS file.
hdu.writeto(output_data, overwrite=True)
hdu.close()
```

2.4.8 imheader

Please review the *Notes* section above before running any examples in this notebook

The imheader task allows the user to list header parameters for a list of images. Here we can use the `astropy` convenience function, `fits.getheader()`. We also show in this example how to save a header to a text file, see the Python file I/O documentation for more details.

```
# Standard Imports
import numpy as np
import glob

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid, productFilename="jczgx1ppq_flc.fits")
obsid = '2004663554'
Observations.download_products(obsid, productFilename="jczgx1ptq_flc.fits")
obsid = '2004663556'
Observations.download_products(obsid, productFilename="jczgx1q1q_flc.fits")

import shutil
shutil.move('./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits','../data/')
shutil.move('./mastDownload/HST/JCZGX1PTQ/jczgx1ptq_flc.fits','../data/')
shutil.move('./mastDownload/HST/JCZGX1Q1Q/jczgx1q1q_flc.fits','../data/')
```

```
# Change these values to your desired data files, glob will capture all wildcard␣
↪matches
test_data = glob.glob('../data/jczgx*')
out_text = 'imheader_out.txt'

for filename in test_data:
    # Pull the header from extension 1 using FITS convenience function.
    # To access multiple header it's better to use the fits.open() function.
    head = fits.getheader(filename, ext=1)

    # Using repr function to format output
    print(repr(head))

    # Save header to text file
    with open(out_text, mode='a') as out_file:
        out_file.write(repr(head))
        out_file.write('\n\n')
```

```
XTENSION= 'IMAGE   '           / IMAGE extension
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2
PCOUNT  =                    0 / required keyword; must = 0
GCOUNT  =                    1 / required keyword; must = 1
ORIGIN  = 'HSTIO/CFITSIO March 2010' / FITS file originator
DATE    = '2017-12-03' / date this file was written (yyyy-mm-dd)
INHERIT =                    T / inherit the primary header
EXTNAME = 'SCI     '           / extension name
EXTVER  =                    1 / extension version number
ROOTNAME= 'jczgx1ppq                         ' / rootname of the observation set
EXPNAME = 'jczgx1ppq               ' / exposure identifier
BUNIT   = 'ELECTRONS'          / brightness units

              / WFC CCD CHIP IDENTIFICATION

CCDCHIP =                    2 / CCD chip (1 or 2)

              / World Coordinate System and Related Parameters

WCSAXES =                    2 / number of World Coordinate System axes
CRPIX1  =               2048.0 / x-coordinate of reference pixel
```

(continues on next page)

(continued from previous page)

```
CRPIX2  =               1024.0 / y-coordinate of reference pixel
CRVAL1  =    127.7729653461655 / first axis value at reference pixel
CRVAL2  =    65.84354161173992 / second axis value at reference pixel
CTYPE1  = 'RA---TAN-SIP'       / the coordinate type for the first axis
CTYPE2  = 'DEC--TAN-SIP'       / the coordinate type for the second axis
CD1_1   = 1.90483532036217E-08 / partial of first axis coordinate w.r.t. x
CD1_2   = -1.3940675227771E-05 / partial of first axis coordinate w.r.t. y
CD2_1   = -1.3846187057971E-05 / partial of second axis coordinate w.r.t. x
CD2_2   = -9.8508094364170E-07 / partial of second axis coordinate w.r.t. y
LTV1    =        0.0000000E+00 / offset in X to subsection start
LTV2    =        0.0000000E+00 / offset in Y to subsection start
RAW_LTV1=                  0.0 / original offset in X to subsection start
RAW_LTV2=                  0.0 / original offset in Y to subsection start
LTM1_1  =                  1.0 / reciprocal of sampling rate in X
LTM2_2  =                  1.0 / reciprocal of sampling rate in Y
ORIENTAT=              -94.0229 / position angle of image y axis (deg. e of n)
RA_APER =   1.277389583333E+02 / RA of aperture reference position
DEC_APER=   6.584194444444E+01 / Declination of aperture reference position
PA_APER =             -94.3071 / Position Angle of reference aperture center (de
VAFACTOR=   1.000063780568E+00 / velocity aberration plate scale factor

              / READOUT DEFINITION PARAMETERS

CENTERA1=                 2073 / subarray axis1 center pt in unbinned dect. pix
CENTERA2=                 1035 / subarray axis2 center pt in unbinned dect. pix
SIZAXIS1=                 4096 / subarray axis1 size in unbinned detector pixels
SIZAXIS2=                 2048 / subarray axis2 size in unbinned detector pixels
BINAXIS1=                    1 / axis1 data bin size in unbinned detector pixels
BINAXIS2=                    1 / axis2 data bin size in unbinned detector pixels

              / PHOTOMETRY KEYWORDS

PHOTMODE= 'ACS WFC1 F814W MJD#57677.0450' / observation con
PHOTFLAM=        7.0486380E-20 / inverse sensitivity, ergs/cm2/Ang/electron
PHOTZPT =       -2.1100000E+01 / ST magnitude zero point
PHOTPLAM=        8.0449937E+03 / Pivot wavelength (Angstroms)
PHOTBW  =        6.5305701E+02 / RMS bandwidth of filter plus detector

              / REPEATED EXPOSURES INFO

NCOMBINE=                    1 / number of image sets combined during CR rejecti

              / DATA PACKET INFORMATION

FILLCNT =                    0 / number of segments containing fill
ERRCNT  =                    0 / number of segments containing errors
PODPSFF =                    F / podps fill present (T/F)
STDCFFF =                    F / science telemetry fill data present (T=1/F=0)
STDCFFP = '0x5569'             / science telemetry fill pattern (hex)

              / ON-BOARD COMPRESSION INFORMATION

WFCMPRSD=                    F / was WFC data compressed? (T/F)
CBLKSIZ =                    0 / size of compression block in 2-byte words
LOSTPIX =                    0 / #pixels lost due to buffer overflow
COMPTYP = 'None    '           / compression type performed (Partial/Full/None)
```

(continues on next page)

(continued from previous page)

```
             / IMAGE STATISTICS AND DATA QUALITY FLAGS

NGOODPIX=              7987438 / number of good pixels
SDQFLAGS=                31743 / serious data quality flags
GOODMIN =       -2.4801433E+02 / minimum value of good pixels
GOODMAX =        9.0880914E+04 / maximum value of good pixels
GOODMEAN=        5.3076767E+01 / mean value of good pixels
SOFTERRS=                    0 / number of soft error pixels (DQF=1)
SNRMIN  =       -7.5930123E+00 / minimum signal to noise of good pixels
SNRMAX  =        2.2929968E+02 / maximum signal to noise of good pixels
SNRMEAN =        5.1801496E+00 / mean value of signal to noise of good pixels
MEANDARK=        6.1097779E+00 / average of the dark values subtracted
MEANBLEV=       -1.3650392E-01 / average of all bias levels subtracted
MEANFLSH=             0.000000 / Mean number of counts in post flash exposure
RADESYS = 'ICRS    '
OCX10   = 0.001964245000000002
OCX11   =  0.04982054148069229
OCY10   =  0.05027000100000004
OCY11   = 0.001500803312490457
IDCSCALE=                 0.05
IDCTHETA=                  0.0
IDCXREF =               2048.0
IDCYREF =               1024.0
IDCV2REF=    257.1520000000001
IDCV3REF=    302.6619900000002
D2IMERR1=  0.04199999943375587 / Maximum error of NPOL correction for axis 1
D2IMDIS1= 'Lookup  '           / Detector to image correction type
D2IM1   = 'EXTVER: 1' / Version number of WCSDVARR extension containing d2im loo
D2IM1   = 'NAXES: 2' / Number of independent variables in d2im function
D2IM1   = 'AXIS.1: 1' / Axis number of the jth independent variable in a d2im fu
D2IM1   = 'AXIS.2: 2' / Axis number of the jth independent variable in a d2im fu
D2IMERR2=  0.06400000303983688 / Maximum error of NPOL correction for axis 2
D2IMDIS2= 'Lookup  '           / Detector to image correction type
D2IM2   = 'EXTVER: 2' / Version number of WCSDVARR extension containing d2im loo
D2IM2   = 'NAXES: 2' / Number of independent variables in d2im function
D2IM2   = 'AXIS.1: 1' / Axis number of the jth independent variable in a d2im fu
D2IM2   = 'AXIS.2: 2' / Axis number of the jth independent variable in a d2im fu
D2IMEXT = 'jref$02c1450oj_d2i.fits'
WCSNAMEO= 'OPUS    '
WCSAXESO=                    2
CRPIX1O =               2100.0
CRPIX2O =               1024.0
CDELT1O =                  1.0
CDELT2O =                  1.0
CUNIT1O = 'deg     '
CUNIT2O = 'deg     '
CTYPE1O = 'RA---TAN'
CTYPE2O = 'DEC--TAN'
CRVAL1O =       127.7729685204
CRVAL2O =       65.84282090734
LONPOLEO=                180.0
LATPOLEO=       65.84282090734
RADESYSO= 'ICRS    '
CD1_1O  =          2.49806E-08
CD1_2O  =         -1.39456E-05
CD2_1O  =         -1.38597E-05
CD2_2O  =         -9.80762E-07
```

(continues on next page)

(continued from previous page)

```
TDDALPHA= ''
TDD_CXA = ''
TDD_CXB =    -1.0658206323E-06
TDD_CTB =     1.5787128139E-06
TDD_CYA = ''
TDD_CYB = ''
TDDBETA = ''
TDD_CTA = ''
IDCTAB  = 'jref$11d1433lj_idc.fits'
A_2_2   = 3.78731328537869E-14
B_0_3   = -3.8365982324508E-10
A_ORDER =                    5
A_0_2   = 2.16316670266357E-06
B_5_0   = -2.9216557962212E-18
A_4_1   = -2.2975314425693E-18
B_3_1   = -9.2662863736411E-16
B_1_1   = 6.18673688121303E-06
A_4_0   = 2.49648430134054E-14
B_2_0   = -1.7485625426539E-06
A_3_2   = 1.79076698558529E-18
B_0_2   = -7.2366916752762E-06
B_2_3   = -4.0303373428367E-19
A_2_1   = -3.3923056140854E-11
B_3_0   = 9.85440944815669E-11
B_ORDER =                    5
A_3_0   = -4.9299373340579E-10
B_2_1   = -5.1770017201658E-10
B_3_2   = -6.5749429811757E-19
A_2_0   = 8.55757690624103E-06
B_0_4   = 4.80879850209643E-15
B_1_3   = 1.17049370338725E-14
A_1_2   = -5.3116725265518E-10
B_0_5   = -3.0673060246341E-17
A_0_5   = 6.02661866571512E-18
A_5_0   = 3.34396903040512E-18
B_4_1   = 1.26957713407563E-18
A_2_3   = 2.16524457164329E-18
A_1_3   = -7.8672443613644E-15
B_2_2   = -2.9754427958761E-14
B_1_4   = 1.23793339962009E-17
B_1_2   = -7.2577430975755E-11
A_1_1   = -5.2167190331715E-06
A_0_4   = 2.30261315411602E-14
B_4_0   = -1.7435196173764E-14
A_3_1   = 6.55120590759313E-15
A_1_4   = -1.4386444581929E-18
A_0_3   = -1.4678926146950E-13
WCSNAME = 'IDC_11d1433lj'
CPERR1  =  0.02756105922162533 / Maximum error of NPOL correction for axis 1
CPDIS1  = 'Lookup  '           / Prior distortion function type
DP1     = 'EXTVER: 1' / Version number of WCSDVARR extension containing lookup d
DP1     = 'NAXES: 2' / Number of independent variables in distortion function
DP1     = 'AXIS.1: 1' / Axis number of the jth independent variable in a distort
DP1     = 'AXIS.2: 2' / Axis number of the jth independent variable in a distort
CPERR2  =  0.01880022883415222 / Maximum error of NPOL correction for axis 2
CPDIS2  = 'Lookup  '           / Prior distortion function type
DP2     = 'EXTVER: 2' / Version number of WCSDVARR extension containing lookup d
```

(continues on next page)

(continued from previous page)

```
DP2     = 'NAXES: 2' / Number of independent variables in distortion function
DP2     = 'AXIS.1: 1' / Axis number of the jth independent variable in a distort
DP2     = 'AXIS.2: 2' / Axis number of the jth independent variable in a distort
NPOLEXT = 'jref$02c1450rj_npl.fits'
MDRIZSKY=    40.54545593261719 / Sky value computed by AstroDrizzle
XTENSION= 'IMAGE   '           / IMAGE extension
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2
PCOUNT  =                    0 / required keyword; must = 0
GCOUNT  =                    1 / required keyword; must = 1
ORIGIN  = 'HSTIO/CFITSIO March 2010' / FITS file originator
DATE    = '2017-12-03' / date this file was written (yyyy-mm-dd)
INHERIT =                    T / inherit the primary header
EXTNAME = 'SCI     '           / extension name
EXTVER  =                    1 / extension version number
ROOTNAME= 'jczgx1ptq                         ' / rootname of the observation set
EXPNAME = 'jczgx1ptq                ' / exposure identifier
BUNIT   = 'ELECTRONS'          / brightness units

              / WFC CCD CHIP IDENTIFICATION

CCDCHIP =                    2 / CCD chip (1 or 2)

              / World Coordinate System and Related Parameters

WCSAXES =                    2 / number of World Coordinate System axes
CRPIX1  =               2048.0 / x-coordinate of reference pixel
CRPIX2  =               1024.0 / y-coordinate of reference pixel
CRVAL1  =     127.774971972961 / first axis value at reference pixel
CRVAL2  =    65.84362363894992 / second axis value at reference pixel
CTYPE1  = 'RA---TAN-SIP'       / the coordinate type for the first axis
CTYPE2  = 'DEC--TAN-SIP'       / the coordinate type for the second axis
CD1_1   = 1.86049319494035E-08 / partial of first axis coordinate w.r.t. x
CD1_2   = -1.3940697878041E-05 / partial of first axis coordinate w.r.t. y
CD2_1   = -1.3846178828081E-05 / partial of second axis coordinate w.r.t. x
CD2_2   = -9.8463386768576E-07 / partial of second axis coordinate w.r.t. y
LTV1    =        0.0000000E+00 / offset in X to subsection start
LTV2    =        0.0000000E+00 / offset in Y to subsection start
RAW_LTV1=                  0.0 / original offset in X to subsection start
RAW_LTV2=                  0.0 / original offset in Y to subsection start
LTM1_1  =                  1.0 / reciprocal of sampling rate in X
LTM2_2  =                  1.0 / reciprocal of sampling rate in Y
ORIENTAT=              -94.021 / position angle of image y axis (deg. e of n)
RA_APER =   1.277409647262E+02 / RA of aperture reference position
DEC_APER=   6.584202691721E+01 / Declination of aperture reference position
PA_APER =             -94.3053 / Position Angle of reference aperture center (de
VAFACTOR=   1.000063143039E+00 / velocity aberration plate scale factor

              / READOUT DEFINITION PARAMETERS

CENTERA1=                 2073 / subarray axis1 center pt in unbinned dect. pix
CENTERA2=                 1035 / subarray axis2 center pt in unbinned dect. pix
SIZAXIS1=                 4096 / subarray axis1 size in unbinned detector pixels
SIZAXIS2=                 2048 / subarray axis2 size in unbinned detector pixels
BINAXIS1=                    1 / axis1 data bin size in unbinned detector pixels
```

(continues on next page)

(continued from previous page)

```
BINAXIS2=                    1 / axis2 data bin size in unbinned detector pixels

              / PHOTOMETRY KEYWORDS

PHOTMODE= 'ACS WFC1 F814W MJD#57677.0536' / observation con
PHOTFLAM=        7.0486380E-20 / inverse sensitivity, ergs/cm2/Ang/electron
PHOTZPT =       -2.1100000E+01 / ST magnitude zero point
PHOTPLAM=        8.0449937E+03 / Pivot wavelength (Angstroms)
PHOTBW  =        6.5305701E+02 / RMS bandwidth of filter plus detector

              / REPEATED EXPOSURES INFO

NCOMBINE=                    1 / number of image sets combined during CR rejecti

              / DATA PACKET INFORMATION

FILLCNT =                    0 / number of segments containing fill
ERRCNT  =                    0 / number of segments containing errors
PODPSFF =                    F / podps fill present (T/F)
STDCFFF =                    F / science telemetry fill data present (T=1/F=0)
STDCFFP = '0x5569'             / science telemetry fill pattern (hex)

              / ON-BOARD COMPRESSION INFORMATION

WFCMPRSD=                    F / was WFC data compressed? (T/F)
CBLKSIZ =                    0 / size of compression block in 2-byte words
LOSTPIX =                    0 / #pixels lost due to buffer overflow
COMPTYP = 'None    '           / compression type performed (Partial/Full/None)

              / IMAGE STATISTICS AND DATA QUALITY FLAGS

NGOODPIX=              7987448 / number of good pixels
SDQFLAGS=                31743 / serious data quality flags
GOODMIN =       -5.6858374E+02 / minimum value of good pixels
GOODMAX =        8.4768180E+04 / maximum value of good pixels
GOODMEAN=        4.5566620E+01 / mean value of good pixels
SOFTERRS=                    0 / number of soft error pixels (DQF=1)
SNRMIN  =       -6.5290461E+00 / minimum signal to noise of good pixels
SNRMAX  =        2.3049573E+02 / maximum signal to noise of good pixels
SNRMEAN =        4.5304279E+00 / mean value of signal to noise of good pixels
MEANDARK=        6.4147372E+00 / average of the dark values subtracted
MEANBLEV=        6.4909774E-01 / average of all bias levels subtracted
MEANFLSH=            0.000000 / Mean number of counts in post flash exposure
RADESYS = 'ICRS    '
OCX10   = 0.001964245000000002
OCX11   =  0.04982054148069229
OCY10   =  0.05027000100000004
OCY11   = 0.001500803312490457
IDCSCALE=                 0.05
IDCTHETA=                  0.0
IDCXREF =               2048.0
IDCYREF =               1024.0
IDCV2REF=    257.1520000000001
IDCV3REF=    302.6619900000002
D2IMERR1=  0.04199999943375587 / Maximum error of NPOL correction for axis 1
D2IMDIS1= 'Lookup  '           / Detector to image correction type
D2IM1   = 'EXTVER: 1' / Version number of WCSDVARR extension containing d2im loo
```

(continues on next page)

(continued from previous page)

```
D2IM1   = 'NAXES: 2' / Number of independent variables in d2im function
D2IM1   = 'AXIS.1: 1' / Axis number of the jth independent variable in a d2im fu
D2IM1   = 'AXIS.2: 2' / Axis number of the jth independent variable in a d2im fu
D2IMERR2=  0.06400000303983688 / Maximum error of NPOL correction for axis 2
D2IMDIS2= 'Lookup  '           / Detector to image correction type
D2IM2   = 'EXTVER: 2' / Version number of WCSDVARR extension containing d2im loo
D2IM2   = 'NAXES: 2' / Number of independent variables in d2im function
D2IM2   = 'AXIS.1: 1' / Axis number of the jth independent variable in a d2im fu
D2IM2   = 'AXIS.2: 2' / Axis number of the jth independent variable in a d2im fu
D2IMEXT = 'jref$02c1450oj_d2i.fits'
WCSNAMEO= 'OPUS    '
WCSAXESO=                    2
CRPIX1O =               2100.0
CRPIX2O =               1024.0
CDELT1O =                  1.0
CDELT2O =                  1.0
CUNIT1O = 'deg     '
CUNIT2O = 'deg     '
CTYPE1O = 'RA---TAN'
CTYPE2O = 'DEC--TAN'
CRVAL1O =       127.7749750908
CRVAL2O =       65.84290293455
LONPOLEO=                180.0
LATPOLEO=       65.84290293455
RADESYSO= 'ICRS    '
CD1_1O  =          2.45367E-08
CD1_2O  =         -1.39456E-05
CD2_1O  =         -1.38597E-05
CD2_2O  = -9.8031499999999E-07
TDDALPHA= ''
TDD_CXA = ''
TDD_CXB =    -1.0658206323E-06
TDD_CTB =     1.5787128139E-06
TDD_CYA = ''
TDD_CYB = ''
TDDBETA = ''
TDD_CTA = ''
IDCTAB  = 'jref$11d1433lj_idc.fits'
A_2_2   = 3.78731328537869E-14
B_0_3   = -3.8365982324508E-10
A_ORDER =                    5
A_0_2   = 2.16316670266357E-06
B_5_0   = -2.9216557962212E-18
A_4_1   = -2.2975314425693E-18
B_3_1   = -9.2662863736411E-16
B_1_1   = 6.18673688121303E-06
A_4_0   = 2.49648430134054E-14
B_2_0   = -1.7485625426539E-06
A_3_2   = 1.79076698558529E-18
B_0_2   = -7.2366916752762E-06
B_2_3   = -4.0303373428367E-19
A_2_1   = -3.3923056140854E-11
B_3_0   = 9.85440944815669E-11
B_ORDER =                    5
A_3_0   = -4.9299373340579E-10
B_2_1   = -5.1770017201658E-10
B_3_2   = -6.5749429811757E-19
```

(continues on next page)

(continued from previous page)

```
A_2_0   = 8.55757690624103E-06
B_0_4   = 4.80879850209643E-15
B_1_3   = 1.17049370338725E-14
A_1_2   = -5.3116725265518E-10
B_0_5   = -3.0673060246341E-17
A_0_5   = 6.02661866571512E-18
A_5_0   = 3.34396903040512E-18
B_4_1   = 1.26957713407563E-18
A_2_3   = 2.16524457164329E-18
A_1_3   = -7.8672443613644E-15
B_2_2   = -2.9754427958761E-14
B_1_4   = 1.23793339962009E-17
B_1_2   = -7.2577430975755E-11
A_1_1   = -5.2167190331715E-06
A_0_4   = 2.30261315411602E-14
B_4_0   = -1.7435196173764E-14
A_3_1   = 6.55120590759313E-15
A_1_4   = -1.4386444581929E-18
A_0_3   = -1.4678926146950E-13
WCSNAME = 'IDC_11d1433lj'
CPERR1  =  0.02756105922162533 / Maximum error of NPOL correction for axis 1
CPDIS1  = 'Lookup  '           / Prior distortion function type
DP1     = 'EXTVER: 1' / Version number of WCSDVARR extension containing lookup d
DP1     = 'NAXES: 2' / Number of independent variables in distortion function
DP1     = 'AXIS.1: 1' / Axis number of the jth independent variable in a distort
DP1     = 'AXIS.2: 2' / Axis number of the jth independent variable in a distort
CPERR2  =  0.01880022883415222 / Maximum error of NPOL correction for axis 2
CPDIS2  = 'Lookup  '           / Prior distortion function type
DP2     = 'EXTVER: 2' / Version number of WCSDVARR extension containing lookup d
DP2     = 'NAXES: 2' / Number of independent variables in distortion function
DP2     = 'AXIS.1: 1' / Axis number of the jth independent variable in a distort
DP2     = 'AXIS.2: 2' / Axis number of the jth independent variable in a distort
NPOLEXT = 'jref$02c1450rj_npl.fits'
MDRIZSKY=    33.60466766357422 / Sky value computed by AstroDrizzle
XTENSION= 'IMAGE   '           / IMAGE extension
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 4096 / length of data axis 1
NAXIS2  =                 2048 / length of data axis 2
PCOUNT  =                    0 / required keyword; must = 0
GCOUNT  =                    1 / required keyword; must = 1
ORIGIN  = 'HSTIO/CFITSIO March 2010' / FITS file originator
DATE    = '2017-12-03' / date this file was written (yyyy-mm-dd)
INHERIT =                    T / inherit the primary header
EXTNAME = 'SCI     '           / extension name
EXTVER  =                    1 / extension version number
ROOTNAME= 'jczgx1q1q                          ' / rootname of the observation set
EXPNAME = 'jczgx1q1q                ' / exposure identifier
BUNIT   = 'ELECTRONS'          / brightness units

              / WFC CCD CHIP IDENTIFICATION

CCDCHIP =                    2 / CCD chip (1 or 2)

              / World Coordinate System and Related Parameters

WCSAXES =                    2 / number of World Coordinate System axes
```

(continues on next page)

(continued from previous page)

```
CRPIX1  =               2048.0 / x-coordinate of reference pixel
CRPIX2  =               1024.0 / y-coordinate of reference pixel
CRVAL1  =    127.7790008405421 / first axis value at reference pixel
CRVAL2  =     65.8438018528099 / second axis value at reference pixel
CTYPE1  = 'RA---TAN-SIP'       / the coordinate type for the first axis
CTYPE2  = 'DEC--TAN-SIP'       / the coordinate type for the second axis
CD1_1   = 1.77165941042396E-08 / partial of first axis coordinate w.r.t. x
CD1_2   = -1.3940911726204E-05 / partial of first axis coordinate w.r.t. y
CD2_1   = -1.3846329672062E-05 / partial of second axis coordinate w.r.t. x
CD2_2   = -9.8374991384276E-07 / partial of second axis coordinate w.r.t. y
LTV1    =        0.0000000E+00 / offset in X to subsection start
LTV2    =        0.0000000E+00 / offset in Y to subsection start
RAW_LTV1=                  0.0 / original offset in X to subsection start
RAW_LTV2=                  0.0 / original offset in Y to subsection start
LTM1_1  =                  1.0 / reciprocal of sampling rate in X
LTM2_2  =                  1.0 / reciprocal of sampling rate in Y
ORIENTAT=             -94.0174 / position angle of image y axis (deg. e of n)
RA_APER =   1.277449931071E+02 / RA of aperture reference position
DEC_APER=   6.584220602391E+01 / Declination of aperture reference position
PA_APER =             -94.3016 / Position Angle of reference aperture center (de
VAFACTOR=   1.000073952797E+00 / velocity aberration plate scale factor

              / READOUT DEFINITION PARAMETERS

CENTERA1=                 2073 / subarray axis1 center pt in unbinned dect. pix
CENTERA2=                 1035 / subarray axis2 center pt in unbinned dect. pix
SIZAXIS1=                 4096 / subarray axis1 size in unbinned detector pixels
SIZAXIS2=                 2048 / subarray axis2 size in unbinned detector pixels
BINAXIS1=                    1 / axis1 data bin size in unbinned detector pixels
BINAXIS2=                    1 / axis2 data bin size in unbinned detector pixels

              / PHOTOMETRY KEYWORDS

PHOTMODE= 'ACS WFC1 F814W MJD#57677.0946' / observation con
PHOTFLAM=        7.0486386E-20 / inverse sensitivity, ergs/cm2/Ang/electron
PHOTZPT =       -2.1100000E+01 / ST magnitude zero point
PHOTPLAM=        8.0449937E+03 / Pivot wavelength (Angstroms)
PHOTBW  =        6.5305701E+02 / RMS bandwidth of filter plus detector

              / REPEATED EXPOSURES INFO

NCOMBINE=                    1 / number of image sets combined during CR rejecti

              / DATA PACKET INFORMATION

FILLCNT =                    0 / number of segments containing fill
ERRCNT  =                    0 / number of segments containing errors
PODPSFF =                    F / podps fill present (T/F)
STDCFFF =                    F / science telemetry fill data present (T=1/F=0)
STDCFFP = '0x5569'             / science telemetry fill pattern (hex)

              / ON-BOARD COMPRESSION INFORMATION

WFCMPRSD=                    F / was WFC data compressed? (T/F)
CBLKSIZ =                    0 / size of compression block in 2-byte words
LOSTPIX =                    0 / #pixels lost due to buffer overflow
COMPTYP = 'None    '           / compression type performed (Partial/Full/None)
```

(continues on next page)

(continued from previous page)

```
              / IMAGE STATISTICS AND DATA QUALITY FLAGS

NGOODPIX=              7987459 / number of good pixels
SDQFLAGS=                31743 / serious data quality flags
GOODMIN =       -4.6811813E+02 / minimum value of good pixels
GOODMAX =        8.6860820E+04 / maximum value of good pixels
GOODMEAN=        5.8565811E+01 / mean value of good pixels
SOFTERRS=                    0 / number of soft error pixels (DQF=1)
SNRMIN  =       -5.3112264E+00 / minimum signal to noise of good pixels
SNRMAX  =        2.3047971E+02 / maximum signal to noise of good pixels
SNRMEAN =        5.8733592E+00 / mean value of signal to noise of good pixels
MEANDARK=        6.1097779E+00 / average of the dark values subtracted
MEANBLEV=       -8.4848583E-01 / average of all bias levels subtracted
MEANFLSH=             0.000000 / Mean number of counts in post flash exposure
RADESYS = 'ICRS    '
OCX10   = 0.001964245000000002
OCX11   =  0.04982054148069229
OCY10   =  0.05027000100000004
OCY11   = 0.001500803312490457
IDCSCALE=                 0.05
IDCTHETA=                  0.0
IDCXREF =               2048.0
IDCYREF =               1024.0
IDCV2REF=    257.1520000000001
IDCV3REF=    302.6619900000002
D2IMERR1=  0.04199999943375587 / Maximum error of NPOL correction for axis 1
D2IMDIS1= 'Lookup  '           / Detector to image correction type
D2IM1   = 'EXTVER: 1' / Version number of WCSDVARR extension containing d2im loo
D2IM1   = 'NAXES: 2' / Number of independent variables in d2im function
D2IM1   = 'AXIS.1: 1' / Axis number of the jth independent variable in a d2im fu
D2IM1   = 'AXIS.2: 2' / Axis number of the jth independent variable in a d2im fu
D2IMERR2=  0.06400000303983688 / Maximum error of NPOL correction for axis 2
D2IMDIS2= 'Lookup  '           / Detector to image correction type
D2IM2   = 'EXTVER: 2' / Version number of WCSDVARR extension containing d2im loo
D2IM2   = 'NAXES: 2' / Number of independent variables in d2im function
D2IM2   = 'AXIS.1: 1' / Axis number of the jth independent variable in a d2im fu
D2IM2   = 'AXIS.2: 2' / Axis number of the jth independent variable in a d2im fu
D2IMEXT = 'jref$02c1450oj_d2i.fits'
WCSNAMEO= 'OPUS    '
WCSAXESO=                    2
CRPIX1O =               2100.0
CRPIX2O =               1024.0
CDELT1O =                  1.0
CDELT2O =                  1.0
CUNIT1O = 'deg     '
CUNIT2O = 'deg     '
CTYPE1O = 'RA---TAN'
CTYPE2O = 'DEC--TAN'
CRVAL1O =       127.7790038454
CRVAL2O =    65.84308114840999
LONPOLEO=                180.0
LATPOLEO=    65.84308114840999
RADESYSO= 'ICRS    '
CD1_1O  =          2.36474E-08
CD1_2O  =         -1.39456E-05
CD2_1O  =         -1.38597E-05
```

(continues on next page)

(continued from previous page)

```
CD2_2O  =          -9.7942E-07
TDDALPHA= ''
TDD_CXA = ''
TDD_CXB =    -1.0658206323E-06
TDD_CTB =     1.5787128139E-06
TDD_CYA = ''
TDD_CYB = ''
TDDBETA = ''
TDD_CTA = ''
IDCTAB  = 'jref$11d1433lj_idc.fits'
A_2_2   = 3.78731328537869E-14
B_0_3   = -3.8365982324508E-10
A_ORDER =                    5
A_0_2   = 2.16316670266357E-06
B_5_0   = -2.9216557962212E-18
A_4_1   = -2.2975314425693E-18
B_3_1   = -9.2662863736411E-16
B_1_1   = 6.18673688121303E-06
A_4_0   = 2.49648430134054E-14
B_2_0   = -1.7485625426539E-06
A_3_2   = 1.79076698558529E-18
B_0_2   = -7.2366916752762E-06
B_2_3   = -4.0303373428367E-19
A_2_1   = -3.3923056140854E-11
B_3_0   = 9.85440944815669E-11
B_ORDER =                    5
A_3_0   = -4.9299373340579E-10
B_2_1   = -5.1770017201658E-10
B_3_2   = -6.5749429811757E-19
A_2_0   = 8.55757690624103E-06
B_0_4   = 4.80879850209643E-15
B_1_3   = 1.17049370338725E-14
A_1_2   = -5.3116725265518E-10
B_0_5   = -3.0673060246341E-17
A_0_5   = 6.02661866571512E-18
A_5_0   = 3.34396903040512E-18
B_4_1   = 1.26957713407563E-18
A_2_3   = 2.16524457164329E-18
A_1_3   = -7.8672443613644E-15
B_2_2   = -2.9754427958761E-14
B_1_4   = 1.23793339962009E-17
B_1_2   = -7.2577430975755E-11
A_1_1   = -5.2167190331715E-06
A_0_4   = 2.30261315411602E-14
B_4_0   = -1.7435196173764E-14
A_3_1   = 6.55120590759313E-15
A_1_4   = -1.4386444581929E-18
A_0_3   = -1.4678926146950E-13
WCSNAME = 'IDC_11d1433lj'
CPERR1  =  0.02756105922162533 / Maximum error of NPOL correction for axis 1
CPDIS1  = 'Lookup  '           / Prior distortion function type
DP1     = 'EXTVER: 1' / Version number of WCSDVARR extension containing lookup d
DP1     = 'NAXES: 2' / Number of independent variables in distortion function
DP1     = 'AXIS.1: 1' / Axis number of the jth independent variable in a distort
DP1     = 'AXIS.2: 2' / Axis number of the jth independent variable in a distort
CPERR2  =  0.01880022883415222 / Maximum error of NPOL correction for axis 2
CPDIS2  = 'Lookup  '           / Prior distortion function type
```

(continues on next page)

(continued from previous page)

```
DP2     = 'EXTVER: 2' / Version number of WCSDVARR extension containing lookup d
DP2     = 'NAXES: 2' / Number of independent variables in distortion function
DP2     = 'AXIS.1: 1' / Axis number of the jth independent variable in a distort
DP2     = 'AXIS.2: 2' / Axis number of the jth independent variable in a distort
NPOLEXT = 'jref$02c1450rj_npl.fits'
MDRIZSKY=    50.43417358398437 / Sky value computed by AstroDrizzle
```

2.4.9 imhistogram

Please review the *Notes* section above before running any examples in this notebook

Imhistogram will plot a customized histogram of the provided image data. To make a histogram in Python we are going to use Matplotlib's `hist` function. See the `hist` documentation for options to change the histogram type, scaling, bin sizes, and more.

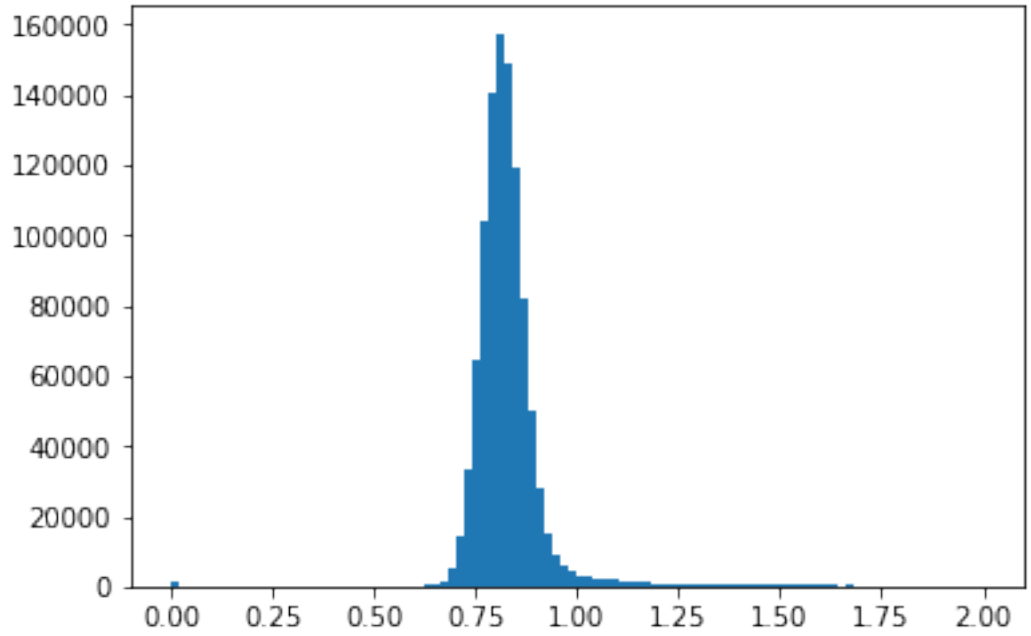
```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Change these values to your desired data files
test_data = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'

# Pull out the first science array, we also need to flatten the data to a
# 1D array before sending it to hist
sci1 = fits.getdata(test_data,ext=1)
sci1f = sci1.flatten()

# Now we can plot our histogram, using some of the optional keywords in hist
# The hist function returns the values of the histogram bins (n), the edges
# of the bins (obins), and the patches used to create the histogram
fig = plt.figure()
n, obins, patches = plt.hist(sci1f,bins=100,range=(0,2))

# Save resulting figure to png file
fig.savefig('hist.png')
```



2.4.10 imreplace

Please review the *Notes* section above before running any examples in this notebook

Imreplace is used to replace array sections with a constant. We can use simple `numpy` array manipulation to replicate imreplace. For details on how to grow the boolean array for replacement see crgrow, or the skimage.dilation documentation.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Change these values to your desired data files
test_data = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
out_file = 'imreplace_out.fits'
```

(continues on next page)

(continued from previous page)

```
# Pull out the first science array
hdu = fits.open(test_data)
sci1 = hdu[1].data

print("cutout of array before replacements:")
print(sci1[50:55, 50:55])

# Make boolean mask with your requirements, here we produce a boolean mask
# where all array elements with values >0.5 and <0.6 are set to True.
mask1 = np.logical_and(sci1>0.8, sci1<0.82)

# Use mask to replace values
sci1[mask1] = 99

print("\ncoutout of array after replacements:")
print(sci1[50:55, 50:55])

# Take updated array and write out new FITS file
hdu[1].data = sci1
hdu.writeto(out_file, overwrite=True)

# Close FITS file
hdu.close()
```

```
cutout of array before replacements:
[[ 0.89118606  0.87640154  0.81239933  0.77495182  0.80048275]
 [ 0.83939391  0.79715788  0.71130604  0.83452195  0.74553812]
 [ 0.82984501  0.82536161  0.82937354  0.82661521  0.80760878]
 [ 0.88277584  0.78050691  0.85906219  0.80846858  0.8092978 ]
 [ 0.85532236  0.73028219  0.81455106  0.76300722  0.85437953]]

coutout of array after replacements:
[[  0.89118606   0.87640154  99.           0.77495182  99.        ]
 [  0.83939391   0.79715788   0.71130604   0.83452195   0.74553812]
 [  0.82984501   0.82536161   0.82937354   0.82661521  99.        ]
 [  0.88277584   0.78050691   0.85906219  99.          99.        ]
 [  0.85532236   0.73028219  99.           0.76300722   0.85437953]]
```

```
# We can also use numpy where to pull out index numbers
mask2 = np.where(sci1 > 1000)
print("Index values where sci1 is > 1,000")
print(mask2)
```

```
Index values where sci1 is > 1,000
(array([ 474,  474,  606,  607,  607,  607,  608,  608,  608,  608,  609,
        609,  609,  609,  610,  610,  610,  804,  804,  809,  809,  810,
        883,  883, 1002, 1013]), array([455, 456, 285, 284, 285, 286, 284, 285, 286,␣
↪287, 284, 285, 286,
       287, 284, 285, 286, 349, 350,  53, 575,  53, 161, 162, 104, 460]))
```

2.4.11 imstack-imslice

Please review the *Notes* section above before running any examples in this notebook

Imstack can take multiple FITS images and stack the data, writing out a new file where the FITS data is 1-dimension higher then the input images. Here we show that manipulation using the `astropy` library and numpy.stack.

Imslice can take a 3-D datacube FITS image and return multiple 2D images sliced through the chosen dimension. Keep in mind for the python equivalent workflow that the header file from the original input image will be used for all output images, including WCS information. We will be using numpy.split.

Below we first produced a 3-D datacube with by stacking, then split the output.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
obsid = '2004663556'
Observations.download_products(obsid, productFilename="jczgx1q1q_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
INFO: Found cached file ./mastDownload/HST/JCZGX1Q1Q/jczgx1q1q_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

Here is an example that stacks arrays into a 3-D datacube

```
# Pull two image data arrays and an image header
header1 = fits.getheader('./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits',ext=1)
image1 = fits.getdata('./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits')
image2 = fits.getdata('./mastDownload/HST/JCZGX1Q1Q/jczgx1q1q_flc.fits')

# Stack arrays, the new dimension will be put first, unless otherwise specified with␣
↪the axis keyword
outstack = np.stack((image1,image2))
print("final shape is:")
print(outstack.shape)

# Now we can write this new array into a new FITS file by packing it back into an HDU␣
↪object
hdu = fits.PrimaryHDU(outstack,header1)
hdu.writeto('imstack_out.fits', overwrite=True)
```

```
final shape is:
(2, 2048, 4096)
```

Now we take that output and break it back down to 2-D arrays.

```
# Pull image data array and image header
orig_hdu = fits.open('imstack_out.fits')

print("Here's the extensions in our input file:")
orig_hdu.info()

header1 = orig_hdu[0].header
image1 = orig_hdu[0].data
orig_hdu.close()

print("\noriginal array - the dimension order is listed " +
      "in reverse order \nnow that we have read the array into a numpy array:")
print(image1.shape)

# Slice images easily by using numpy.split, which returns a list of the output arrays
# THen numpy.squeeze is used to remove the extra length one dimensions left over from
# numpy.split.
arr_list = np.split(image1, 2)
arr_list = np.squeeze(arr_list)
print("\nfinal shape of a slice is:")
print(arr_list[0].shape)

# Now we can write this new array into a new FITS files by packing it back into an␣
↪HDU object
hdu1 = fits.PrimaryHDU(arr_list[0],header1)
hdu1.writeto('imslice_out1.fits', overwrite=True)
hdu2 = fits.PrimaryHDU(arr_list[1],header1)
hdu2.writeto('imslice_out2.fits', overwrite=True)
```

```
Here's the extensions in our input file:
Filename: imstack_out.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  SCI           1 PrimaryHDU     199   (4096, 2048, 2)   float32

original array - the dimension order is listed in reverse order
now that we have read the array into a numpy array:
(2, 2048, 4096)

final shape of a slice is:
(2048, 4096)
```

2.4.12 imstatistics

Please review the *Notes* section above before running any examples in this notebook

We will use the `astropy.stats.sigma_clipped_stats` function here, which has some wider capabilites then the imstatistics function. Please see the `stats` package documentation for details on the advanced usage. We also use some Numpy functions for additional statistics.

Important Note to Users: There are some small differences in algorithms between the IRAF and Python statistical calculations. Both the centering function used for the clipping as well as the degrees of freedom may vary. For example, `astropy.stats.sigma_clipped_stats` uses a simple median for the clipping center. However a custom centering function can be provided through the `cenfunc` parameter. Proceed with care if you are comparing prior IRAF results to Python results.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astropy import stats
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Change these values to your desired data files
test_data = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
sci1 = fits.getdata(test_data, ext=1)

# The sigma_clipped_stats function returns the mean, median, and stddev respectively
# To more closely replicate the IRAF version that is using n-1 in it's calculations
# we use the std_ddof parameter
output = stats.sigma_clipped_stats(sci1, sigma=3.0, iters=3, std_ddof=1)
print("mean, median, standard deviation:")
print(output)

# To see the min and max of an array we can use numpy.min and numpy.max
array_min = np.min(sci1)
array_max = np.max(sci1)
print("\nmin, max")
print("{}, {}".format(array_min, array_max))

# To find out how many pixels are greater then a particular value we can use numpy.
↪where
where_result = np.where(sci1 > 1000)
count = len(where_result[0])
print("\nNumber of pixels above 1,000:")
print(count)
```

```
mean, median, standard deviation:
(0.82595410841884809, 0.81768394, 0.074634554991261454)

min, max
-4007.712890625, 27569.6015625

Number of pixels above 1,000:
26
```

2.4.13 imsum

Please review the *Notes* section above before running any examples in this notebook

Imsum is used to compute the sum, average, or mean of a set of images. We will be using the `ccdproc Combiner` class here. Keep in mind that the original FITS header is not retained in the `CCDData` object. Please see the ccdproc documentation for more details.

```
# Astronomy Specific Imports
from astropy.io import fits
from astropy import units
from ccdproc import CCDData, Combiner
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615003'
Observations.download_products(obsid,productFilename="iczgs3y5q_flt.fits")
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Change these values to your desired data files
test_data1 = './mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits'
test_data2 = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'

# First we need to pull out the science arrays to create CCDData objects
# Our actual unit is electrons/sec, this is not accepted by the current
# set of units
cdata1 = CCDData.read(test_data1, hdu=1, unit=units.electron/units.s)
cdata2 = cdata1.copy()
cdata3 = CCDData.read(test_data2, hdu=1, unit=units.electron/units.s)
cdata4 = cdata3.copy()
combiner = Combiner([cdata1, cdata2, cdata3, cdata4])

# Now we can make our mask for extrema clipping
# The equivalent of low_reject, high_reject parameter
combiner.clip_extrema(nlow=1, nhigh=1)

# And finally to combine...
final_combine = combiner.average_combine()
print(final_combine.data)
```

```
INFO:astropy:using the unit electron / s passed to the FITS reader instead of the␣
↪unit ELECTRONS/S in the FITS file.
INFO:astropy:using the unit electron / s passed to the FITS reader instead of the␣
↪unit ELECTRONS/S in the FITS file.
```

```
INFO: using the unit electron / s passed to the FITS reader instead of the unit␣
↪ELECTRONS/S in the FITS file. [astropy.nddata.ccddata]
INFO: using the unit electron / s passed to the FITS reader instead of the unit␣
↪ELECTRONS/S in the FITS file. [astropy.nddata.ccddata]
[[  0.87720111   0.82106587   0.79521415 ...,   3.87308204   7.41545987
    9.01969481]
 [  0.89028609   0.7884455    0.8240625  ...,   0.86163342   4.53510189
    0.99109203]
 [  0.81683022   0.83273572   0.82175627 ...,   3.60699821  -7.82266164
    2.95994186]
 ...,
 [ 40.72796059  15.36561799  -8.79329443 ...,  22.68277168  25.31048012
   28.829813  ]
 [ 46.28870392  -4.50218874   1.74757147 ...,  13.24364138  25.70440292
   11.0971849 ]
 [ 42.8106432   29.66250706  63.18441772 ...,   0.           9.80057049
   22.66858006]]
```

2.4.14 listpixels

Please review the *Notes* section above before running any examples in this notebook

Listpixels was used to list an indexed section of a FITS data array. This is easy to do using `astropy`, but **keep in mind that Python indexes from zero, and with the y-axis leading, i.e. [y,x]**. You also want to end the cut with the pixel *after* the end pixel. So to get 1-10 in x and 5-15 in y, you will index like so: array[4:15,0:10]. To see listpixels results for more then one file, you will need to loop over a list of files, see information about Python loops here.

```
# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615003'
Observations.download_products(obsid,productFilename="iczgs3y5q_flt.fits")
```

```
INFO:astropy:Found cached file ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits with␣
↪expected size 16534080.
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Change this value to your desired data files
test_data1 = './mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits'

# To quickly pull out the data array you can use the astropy convenience function
data_arr = fits.getdata(test_data1,ext=1)
```

(continues on next page)

(continued from previous page)

```
# Now we can index the array as desired
# We're cutting out 5 in y, and 2 in x
print(data_arr[0:5,0:2])
```

```
[[ 0.86692303  0.80678135]
 [ 0.83312052  0.76854318]
 [ 0.77341086  0.80276382]
 [ 0.80539584  0.78261763]
 [ 0.78274417  0.82206035]]
```

2.4.15 Not Replacing

- imrename - can use command line utilities or the Python `os` package for this functionality.
- imdelete - can use command line utilities or the Python `os` package for this functionality.
- imtile - **may** replace infuture
- sections - IRAF utility function
- imgets - see *images.imutil.hselect*
- minmax - see *images.imutil.imstatistics*

2.5 images.tv

The tv package contains interactive image display utilities.

2.5.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Many of the images.tv functionality has been replaced by the Python imexam package, which works in conjunction with both the ginga and ds9 viewers. In this notebook we will simply provide the task name for imexam that covers the IRAF task. We leave it to the user to reference the imexam documentation for further information.

Contents:

2.5.2 display

Please review the *Notes* section above before running any examples in this notebook

display - Load an image or image section into the display

- see http://imexam.readthedocs.io/en/latest/imexam/examples.html#basic-usage and http://imexam.readthedocs.io/en/latest/imexam/imexam_command.html#cutout-a-simple-fits-image

2.5.3 imexamine

Please review the *Notes* section above before running any examples in this notebook

imexamine - Examine images using image display, graphics, and text

- see http://imexam.readthedocs.io/en/latest/imexam/iraf_imexam.html

2.5.4 bpmedit-imedit-badpixcorr

Please review the *Notes* section above before running any examples in this notebook

bpmedit-badpixcorr - examine and edit bad pixel masks associated with images (badpixcorr in ginga)

- see http://stginga.readthedocs.io/en/latest/stginga/plugins_manual/badpixcorr.html

imedit - Examine and edit pixels in images

- see http://imexam.readthedocs.io/en/v0.7.1/imexam/imexam_command.html#pixel-coordinates-and-value, combine with astropy.io.fits

2.5.5 tvmark

Please review the *Notes* section above before running any examples in this notebook

tvmark - Mark objects on the image display

- see http://ginga.readthedocs.io/en/latest/manual/plugins_local/tvmark.html

2.5.6 wcslab

Please review the *Notes* section above before running any examples in this notebook

wcslab - Overlay a displayed image with a world coordinate grid.

- see http://ginga.readthedocs.io/en/latest/manual/plugins_local/wcsaxes.html?highlight=wcs

2.5.7 Not Replacing

- iis subpackage - these utilities are replaced by ginga and ds9.

2.6 stsdas.analysis.fitting

Curve fitting tools.

2.6.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

The stsdas.analysis.fitting package is used to do various types of 1D and 2D fitting. Fitting has been well developed in Python with the Scipy and Astropy libraries. We have covered these tasks in other IRAF notebooks, and refer to those entries below.

Contents:

2.6.2 gfit1d-nfit1d-ngaussfit

Please review the *Notes* **section above before running any examples in this notebook**

Gfit1d and nfit1d are used to interactively achieve a 1-d fit to an image, tables or lists. For various interactive curve fitting on images please see the Python imexam. For non-interactive fitting of a table or list, please see the tasks in the **tables.ttools** notebook, and the **images.imfit.fit1d-lineclean** tasks.

2.6.3 function

Please review the *Notes* **section above before running any examples in this notebook**

The function task is used to apply functions to images, tables or lists. For image function application see **images.imutil.imfunction-imexpr**. Once the table or list is coverted to a numpy array, the method for applying a fucntion is the same as **images.imutil.imfunction-imexpr**. See **tables.ttools.taextract-tainsert** for table conversions.

2.6.4 Not Replacing

- controlpars - Pset with algorithm control parameters. Deprecated.
- errorpars - Pset with error-related parameters. Deprecated.
- bbodypars - Pset with parameters for black-body function. Deprecated.
- cgausspars - Pset with parameters for constrained Gaussians function. Deprecated.
- comppars - Pset with parameters for composite (bb + powerlaw) function. Deprecated.
- galprofpars - Pset with parameters for galaxy profile function. Deprecated.
- gausspars - Pset with parameters for Gaussians function. Deprecated.
- i2gaussfit - Iterative 2-d Gaussian fit to noisy images. Deprecated.
- n2gaussfit - 2-d Gaussian fit to images. See **images.imfit.imsurfit**
- powerpars - Pset with parameters for powerlaw function. Deprecated.

- prfit - Print contents of fit tables created by fitting task. Deprecated.
- samplepars - Pset with data sampling parameters. Deprecated.
- tgausspars - Pset with parameters for two-dim Gaussian function. Deprecated.
- twobbpars - Pset with parameters for two-black-body function. Deprecated.
- userpars - Pset with parameters for user-defined function. Deprecated.

2.7 stsdas.toolbox.imgtools

Tasks for performing operations on images and masks.

2.7.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

The various image tasks found in the stsdas.toolbox.imgtools package have been replaced in the Numpy and Astropy libraries.

Contents:

2.7.2 addmasks

Please review the *Notes* **section above before running any examples in this notebook**

Addmasks is used to combine several masks or bad pixel lists. We can do this using the `numpy` bitwise tasks: bitwise_or, bitwise_and, and invert, along with a slew of other numpy bit functions. Below we show examples of `bitwise_and` and `bitwise_or`.

```
# Standard Imports
import numpy as np
```

```
a = np.array([1,4,10])
b = np.array([1,0,8])

# OR
print(np.bitwise_or(a,b))

# AND
print(np.bitwise_and(a,b))
```

```
[ 1  4 10]
[1 0 8]
```

2.7.3 iminsert

Please review the *Notes* section above before running any examples in this notebook

Iminsert is used to insert a small image into a larger image. This is easy to do with the Numpy array indexing after you've read in your images with `Astropy.io.fits`. Below we'll show a quick array example.

```
# Standard Imports
import numpy as np
```

```
# generate test arrays
my_array = np.random.rand(7,5)
ones = np.array(([1,1],[1,1]))

# replace middle 3x3 square with ones
my_array[2:4,2:4] = ones

# Print result
print(my_array)
```

```
[[ 0.06888833  0.15088263  0.00241     0.09282496  0.07325408]
 [ 0.78665832  0.3402431   0.5265134   0.46253075  0.54305974]
 [ 0.63473001  0.92986634  1.          1.          0.0904689 ]
 [ 0.2887482   0.50178461  1.          1.          0.78550679]
 [ 0.07945175  0.12885675  0.06588469  0.63534732  0.62024358]
 [ 0.53344071  0.2852475   0.03736071  0.30043438  0.97523821]
 [ 0.10331126  0.52996828  0.51318396  0.47988347  0.7098808 ]]
```

2.7.4 improject

Please review the *Notes* section above before running any examples in this notebook

Improject is used to sum or average an image along one axis. This can be accomplised using the numpy.average or the numpy.sum functions and choosing which dimensions you wish to collapse. Below we show an example using `numpy.average`.

```
# Standard Imports
import numpy as np
```

```
# build random test array
my_array = np.random.rand(5,4,3)

# reduce third dimension down
new_array = np.average(my_array, axis=2)
print(new_array.shape)
print(new_array)

# reduce second dimension down
new_array_2 = np.average(my_array, axis=1)
print(new_array_2.shape)
print(new_array_2)
```

```
(5, 4)
[[ 0.60660306  0.55628564  0.79297796  0.73016308]
 [ 0.48911929  0.36071454  0.6167648   0.4261005 ]
 [ 0.47187441  0.21748297  0.92223167  0.64068855]
 [ 0.14900289  0.70091688  0.51759779  0.29799824]
 [ 0.85235487  0.79360714  0.60374945  0.40032384]]
(5, 3)
[[ 0.70389997  0.59038403  0.72023831]
 [ 0.4937127   0.44684555  0.47896609]
 [ 0.43435416  0.5368765   0.71797754]
 [ 0.45942245  0.4114324   0.37828199]
 [ 0.64567646  0.51639255  0.82545747]]
```

2.7.5 mkgauss

Please review the *Notes* section above before running any examples in this notebook

The mkgauss funtionality has been replicated in the Photutils package with photutils.datasets.make_random_gaussians_table and photutils.datasets.make_gaussian_sources_image.

2.7.6 pixlocate

Please review the *Notes* section above before running any examples in this notebook

Pixlocate is used to print positions matching a certain value condition. This is replicated with the `numpy.where` function. Please see the documentation for more details and examples.

2.7.7 rd2xy-xy2rd

Please review the *Notes* section above before running any examples in this notebook

Rd2xy and xy2rd are used to translate RA/Dec to the pixel coordinate and vice-versa. This capability is well covered in the `astropy.wcs` package. Please see the documentation for more details on usage.

2.7.8 Not Replacing

- boxinterp - Fill areas with smoothed values from surrounding area. See **images.imfit** notebook.
- countfiles - Count how many files are in the input file template. Deprecated.
- gcombine - Combine a set of GEIS images into one image. Deprecated, for FITS see **stsdas.toolbox.imgtools.mstools.mscombine**
- gcopy - Generic multi-group copy utility. GEIS, deprecated.
- gstatistics - Compute and print image pixel statistics for all groups. GEIS, deprecated. For FITS see **images.imutil.imstatistics**
- imcalc - Perform general arithmetic operations on images. See **images.imtuil.imarith**.
- imfill - Set fill value in image according to a mask. See **images.imutil.imreplace**.
- listarea - Print an area of an image. See numpy basics documentation.
- moveheader - Combine the header and pixels from two images. GEIS, deprecated.

- pickfile - Get the file name picked from the input file template. Deprecated.
- pixedit - Screen editor for image pixels. See **images.tv.imedit**
- rbinary - Create an image from a binary file. Deprecated.
- stack - Stack images to form a new image with one more dimension. See **images.imutil.imstack**
- xyztable - Interpolate table values, writing results to a table. See **images.imfit.imsurfit** and **tables.ttools.tcopy-tdump**
- xyztoim - Interpolate table values, writing results to an image. See **images.imfit.imsurfit**, Astropy Tables documentation, and **tables.ttools.tcopy-tdump**.

2.8 stsdas.toolbox.imgtools.mstools

Tasks to handle HST imsets.

2.8.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

The imgtools.mstools package contains tasks for working with STIS, NICMOS, ACS, and WFC3 data. Some tasks are 'extensions" of existing tasks in the STSDAS system, and support other instruments/file formats as well.

Contents:

2.8.2 ecdel-ecextract-extdel-msdel-msjoin-mssplit

Please review the *Notes* section above before running any examples in this notebook

These tasks contain various methods for deleting or moving extensions around in FITS files. This can be easily done using the `astropy.io.fits` module in Astropy. Here is a good page to familarize yourself with this package.

2.8.3 mscombine

Please review the *Notes* section above before running any examples in this notebook

The original `mscombine` IRAF task performed image combination of several `SCI` exetensions of HST data while allowing the user to reject specified `DQ` bits. Additionally, the user could choose to combine the stack using the average or the median. This was similar to the `imcombine` task. This example can be used to replicate either task.

This `mscombine` alternative uses `numpy` masked arrays to avoid using flagged pixels in the `DQ` array. In this simple example, we average-combine several full-frame WFC3/UVIS images.

Tasks for image combination are currently being developed in the `CCDPROC` package, see the CCDPROC doc page for more details or the **images.imutil.imsum** task for a short usage example.

```
# Standard Imports
import glob
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from stsci.tools.bitmask import bitfield_to_boolean_mask
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid, productFilename="jczgx1ppq_flc.fits")
obsid = '2004663554'
Observations.download_products(obsid, productFilename="jczgx1ptq_flc.fits")
obsid = '2004663556'
Observations.download_products(obsid, productFilename="jczgx1q1q_flc.fits")

import shutil
shutil.move('./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits','../data/')
shutil.move('./mastDownload/HST/JCZGX1PTQ/jczgx1ptq_flc.fits','../data/')
shutil.move('./mastDownload/HST/JCZGX1Q1Q/jczgx1q1q_flc.fits','../data/')
```

```
Downloading URL https://mast.stsci.edu/api/v0/download/file?uri=mast:HST/product/
↪jczgx1ppq/jczgx1ppq_flc.fits to ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits ...␣
↪[Done]
Downloading URL https://mast.stsci.edu/api/v0/download/file?uri=mast:HST/product/
↪jczgx1ptq/jczgx1ptq_flc.fits to ./mastDownload/HST/JCZGX1PTQ/jczgx1ptq_flc.fits ...␣
↪[Done]
Downloading URL https://mast.stsci.edu/api/v0/download/file?uri=mast:HST/product/
↪jczgx1q1q/jczgx1q1q_flc.fits to ./mastDownload/HST/JCZGX1Q1Q/jczgx1q1q_flc.fits ...␣
↪[Done]
```

```
'../data/jczgx1q1q_flc.fits'
```

```
# Get the data
test_data = glob.glob('../data/jcz*flc.fits')
```

```
# Create masked arrays
masked_arrays_ext1, masked_arrays_ext2, masked_arrays_ext4, masked_arrays_ext5 = [],␣
↪[], [], []
for filename in test_data:
    with fits.open(filename) as hdulist:

        # For UVIS chip 2, using DQ flags 32 and 64 (96 bitflag)
        mask_ext3 = np.bitwise_and(hdulist[3].data, 96) != 0
        masked_arrays_ext1.append(np.ma.masked_array(hdulist[1].data, mask=mask_ext3))
        masked_arrays_ext2.append(np.ma.masked_array(hdulist[2].data, mask=mask_ext3))

        # For UVIS chip 1
        mask_ext6 = np.bitwise_and(hdulist[6].data, 96) != 0
        masked_arrays_ext4.append(np.ma.masked_array(hdulist[4].data, mask=mask_ext6))
        masked_arrays_ext5.append(np.ma.masked_array(hdulist[5].data, mask=mask_ext6))
```

```
# Average-combine SCI arrays
comb_ext1 = np.ma.mean(masked_arrays_ext1, axis=0).data
comb_ext4 = np.ma.mean(masked_arrays_ext4, axis=0).data
```

```
# Propoagate uncertainties for ERR arrays, divide by zero expected
weight_image_ext1 = np.zeros((2048, 4096))
weight_image_ext4 = np.zeros((2048, 4096))
for array in masked_arrays_ext1:
    mask = array.mask
    weight_image_ext1[np.where(mask == False)] += 1.0
for array in masked_arrays_ext4:
    mask = array.mask
    weight_image_ext4[np.where(mask == False)] += 1.0
masked_arrays_ext2_squared = [(item * (1/weight_image_ext1))**2 for item in masked_
↪arrays_ext2]
masked_arrays_ext5_squared = [(item * (1/weight_image_ext4))**2 for item in masked_
↪arrays_ext5]
comb_ext2 = np.sqrt(np.ma.sum(masked_arrays_ext2_squared, axis=0)).data
comb_ext5 = np.sqrt(np.ma.sum(masked_arrays_ext5_squared, axis=0)).data
```

```
/Users/ogaz/miniconda3/envs/irafdev/lib/python3.5/site-packages/ipykernel_launcher.
↪py:10: RuntimeWarning: divide by zero encountered in true_divide
  # Remove the CWD from sys.path while we load stuff.
/Users/ogaz/miniconda3/envs/irafdev/lib/python3.5/site-packages/ipykernel_launcher.
↪py:11: RuntimeWarning: divide by zero encountered in true_divide
  # This is added back by InteractiveShellApp.init_path()
```

```
# Create empty DQ arrays
comb_ext3 = np.zeros((2048, 4096))
comb_ext6 = np.zeros((2048, 4096))
```

```
# Build and save the combined file, using the first final for the header
hdu0 = fits.PrimaryHDU(header=fits.getheader(test_data[0], 0))
hdu1 = fits.ImageHDU(comb_ext1, header=fits.getheader(test_data[0], 0))
hdu2 = fits.ImageHDU(comb_ext2, header=fits.getheader(test_data[0], 1))
hdu3 = fits.ImageHDU(comb_ext3, header=fits.getheader(test_data[0], 2))
hdu4 = fits.ImageHDU(comb_ext4, header=fits.getheader(test_data[0], 3))
hdu5 = fits.ImageHDU(comb_ext5, header=fits.getheader(test_data[0], 4))
hdu6 = fits.ImageHDU(comb_ext6, header=fits.getheader(test_data[0], 5))
hdulist = fits.HDUList([hdu0, hdu1, hdu2, hdu3, hdu4, hdu5, hdu6])
hdulist.writeto('mscombine_test.fits', overwrite=True)
```

2.8.4 msstatistics

Please review the *Notes* section above before running any examples in this notebook

The msstatictics task is similar to images.imutil.imstatistics, but with the added capability to mask using an HST DQ array. Below we show an example of this using multiple files and the sigma_clipped_stats function. For more examples on array statistics please see the images.imutil.imstatistics notebook entry.

```
# Standard Imports
import glob
import numpy as np

```

(continues on next page)

(continued from previous page)

```
# Astronomy Specific Imports
from astropy.io import fits
from astropy import stats
```

```
# Change these values to your desired data file list
# loop over multiple files, make filelist
test_files = glob.glob('../data/n*_tmp.fits')

for filename in test_files:
    hdulist = fits.open(filename)

    # Make mask using Python bitmath, using bit flags 32 and 4
    # Add the values of the flags you would like to mask, and use
    # that value in the np.bitwise_and call.
    boolean_mask = np.bitwise_and(hdulist[3].data, 36) != 0

    # The sigma_clipped_stats function returns the mean, median, and stddev␣
↪respectively
    mean, median, std = stats.sigma_clipped_stats(hdulist[1].data, mask=boolean_mask,␣
↪sigma=2.0, iters=3)
    print("Stats for file: {}".format(filename))
    print("mean: {}".format(mean))
    print("median: {}".format(median))
    print("standard deviation: {}\n".format(std))

    # Close fits file
    hdulist.close()
```

```
Stats for file: ../data/nnicqr34r1q_blv_tmp.fits
mean: 1.049938712724799
median: 0.8347640037536621
standard deviation: 3.386821124737488

Stats for file: ../data/nnicqr34rgq_blv_tmp.fits
mean: 1.0696971193430191
median: 0.8951225280761719
standard deviation: 3.341097790698396

Stats for file: ../data/nnicqr34rvq_blv_tmp.fits
mean: 1.036385163417633
median: 0.8546183109283447
standard deviation: 3.405510574506165
```

2.8.5 Not Replacing

- msarith - Image arithmetic with NICMOS and STIS files. See **images.imutil.imarith**.
- mscopy - Copy image sets of a multi-extension FITS file. See **images.imutil.imcopy**
- mssort - Sort a FITS file to get all extensions of like version number. Deprecated.

Fits Tools:

2.9 fitsutil

General fits file utilities.

2.9.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

For the compression tasks included in fitsutil, astropy has replaced this functionality with the CompImageHDU class. We list both compression tasks together in this notebook with a few examples to show the usage of `CompImageHDU`. the astropy.io.fits can natively open compressed file with a standard `fits.open` command. To uncompress the file, you can then save the fits file object out to a new file.

`astropy.io.fits` is the library responsible for opening and closing fits files. It opens the file into a `HDUList` object, which contains multiple `HDU` objects that can be indexed with integers. Each `HDU` object contains array data object, and a header object.

Contents:

2.9.2 fpack-ricepack

Please review the *Notes* **section above before running any examples in this notebook**

We can compress an image HDU using the fits.CompImageHDU class in `astropy`. This class has several compression options (RICE, PLIO, GZIP, and HCOMPRESS). Here we show one example using RICE compression and another using GZIP compression

```
# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
# RICE example

# test files
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
outfile_rice = 'jczgx1ppq_rice.fits'

# open FITS file
hdulist = fits.open(test_data)

# print HDUList info
hdulist.info()

# compress and save to output file
hdu_rice = fits.CompImageHDU(data=hdulist[1].data, header=hdulist[1].header,␣
↪compression_type='RICE_1')
hdulist_rice = fits.HDUList([hdulist[0],hdu_rice])
hdulist_rice.writeto(outfile_rice, overwrite=True)
hdulist.close()
```

```
Filename: ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     279   ()
  1  SCI           1 ImageHDU       200   (4096, 2048)   float32
  2  ERR           1 ImageHDU        56   (4096, 2048)   float32
  3  DQ            1 ImageHDU        48   (4096, 2048)   int16
  4  SCI           2 ImageHDU       198   (4096, 2048)   float32
  5  ERR           2 ImageHDU        56   (4096, 2048)   float32
  6  DQ            2 ImageHDU        48   (4096, 2048)   int16
  7  D2IMARR       1 ImageHDU        15   (64, 32)   float32
  8  D2IMARR       2 ImageHDU        15   (64, 32)   float32
  9  D2IMARR       3 ImageHDU        15   (64, 32)   float32
 10  D2IMARR       4 ImageHDU        15   (64, 32)   float32
 11  WCSDVARR      1 ImageHDU        15   (64, 32)   float32
 12  WCSDVARR      2 ImageHDU        15   (64, 32)   float32
 13  WCSDVARR      3 ImageHDU        15   (64, 32)   float32
 14  WCSDVARR      4 ImageHDU        15   (64, 32)   float32
 15  WCSCORR       1 BinTableHDU     59   14R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
```

```
# Gzip example

# test files
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
outfile_gzip = 'jczgx1ppq_gzip.fits'

hdulist = fits.open(test_data)
hdu_gzip = fits.CompImageHDU(data=hdulist[1].data, header=hdulist[1].header,␣
↪compression_type='GZIP_1')
hdulist_gzip = fits.HDUList([hdulist[0],hdu_gzip])
hdulist_gzip.writeto(outfile_gzip, overwrite=True)
hdulist.close()
```

2.9.3 fxcopy-fxinsert

Please review the *Notes* section above before running any examples in this notebook

Here we show how to copy out and add new HDU objects, the `astropy` equivalent of fxcopy and fxinsert.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
# test files
test_data = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'
outfile = 'fxinsert.fits'

# open fits file, this outputs an hdulist object
hdulist = fits.open(test_data)

print("hdulist before:")
hdulist.info()

# now let's pull out a reference (copy) of an HDU object from this HDUList
my_hdu = hdulist[1]

# Now let's create a new array to make a new HDU object, this will be the primary HDU
new = np.arange(100.0)
new_hdu = fits.PrimaryHDU(new)

# Now we can create a new HDUList object to put our HDU objects into
my_hdulist = fits.HDUList([new_hdu,my_hdu])

print("\n new hdulist:")
my_hdulist.info()

# Now we close write our new HDUList to a file, and close our test_data file
my_hdulist.writeto(outfile, overwrite=True)
hdulist.close()
```

```
hdulist before:
Filename: ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     279   ()
  1  SCI           1 ImageHDU       200   (4096, 2048)   float32
  2  ERR           1 ImageHDU        56   (4096, 2048)   float32
  3  DQ            1 ImageHDU        48   (4096, 2048)   int16
  4  SCI           2 ImageHDU       198   (4096, 2048)   float32
```

(continues on next page)

(continued from previous page)

```
  5  ERR           2 ImageHDU        56   (4096, 2048)   float32
  6  DQ            2 ImageHDU        48   (4096, 2048)   int16
  7  D2IMARR       1 ImageHDU        15   (64, 32)   float32
  8  D2IMARR       2 ImageHDU        15   (64, 32)   float32
  9  D2IMARR       3 ImageHDU        15   (64, 32)   float32
 10  D2IMARR       4 ImageHDU        15   (64, 32)   float32
 11  WCSDVARR      1 ImageHDU        15   (64, 32)   float32
 12  WCSDVARR      2 ImageHDU        15   (64, 32)   float32
 13  WCSDVARR      3 ImageHDU        15   (64, 32)   float32
 14  WCSDVARR      4 ImageHDU        15   (64, 32)   float32
 15  WCSCORR       1 BinTableHDU     59   14R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]

 new hdulist:
Filename: (No file associated with this HDUList)
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU       5   (100,)   float64
  1  SCI           1 ImageHDU       200   (4096, 2048)   float32
```

2.9.4 fxdelete-fxsplit-fxextract

Please review the *Notes* section above before running any examples in this notebook

fxdelete will delete a FITS extension in place, and fxsplit and fxextract will take a multiple extension FITS file and break them out into single FITS files. Both these tasks can be done using astropy.io.fits. Below we show some a short example. We will pull out the 3rd extension from the test file, save it to a new fits file, and delete that extension from the original `HDUList`

```
# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615003'
Observations.download_products(obsid,productFilename="iczgs3y5q_flt.fits")
```

```
Downloading URL https://mast.stsci.edu/api/v0/download/file?uri=mast:HST/product/
↪iczgs3y5q/iczgs3y5q_flt.fits to ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits ...␣
↪[Done]
```

```
# FITS filenames
test_data = './mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits'
outfile_1 = 'fxsplit.fits'
outfile_2 = 'fxdelete.fits'

# Print out some stats for this file
print("original FITS file:")
fits.info(test_data)

# Open FITS file
hdulist = fits.open(test_data)
```

(continues on next page)

(continued from previous page)

```
# Pull out single HDU extension and put into new FITS file
single_HDU = hdulist[3]
primary_HDU = fits.PrimaryHDU()
new_hdulist = fits.HDUList([primary_HDU,single_HDU])
print("\n\nnew FITS file with just the 3rd extension:")
new_hdulist.info()
new_hdulist.writeto(outfile_1, overwrite=True)


# Now save a new copy of the original file without that third extension
edited_hdulist = fits.HDUList([hdulist[0],hdulist[1],hdulist[2],hdulist[4],hdulist[5],
↪hdulist[6]])
print(type(hdulist))
print("\n\nnew FITS file with the 3rd extension taken out:")
edited_hdulist.info()
edited_hdulist.writeto(outfile_2, overwrite=True)

# Close original file
hdulist.close()
```

```
original FITS file:
Filename: ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     265   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
  3  DQ            1 ImageHDU        43   (1014, 1014)   int16
  4  SAMP          1 ImageHDU        37   (1014, 1014)   int16
  5  TIME          1 ImageHDU        37   (1014, 1014)   float32
  6  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]


new FITS file with just the 3rd extension:
Filename: (No file associated with this HDUList)
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU       4   ()
  1  DQ            1 ImageHDU        43   (1014, 1014)   int16
<class 'astropy.io.fits.hdu.hdulist.HDUList'>


new FITS file with the 3rd extension taken out:
Filename: (No file associated with this HDUList)
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     265   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
  3  SAMP          1 ImageHDU        37   (1014, 1014)   int16
  4  TIME          1 ImageHDU        37   (1014, 1014)   float32
  5  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
```

2.9.5 fxdummyh

Please review the *Notes* section above before running any examples in this notebook

Fxdummyh will create an empty fits file.

```
# Astronomy Specific Imports
from astropy.io import fits
```

```
# Write empty file
hdup = fits.PrimaryHDU()
hdu1 = fits.ImageHDU()
hdu2 = fits.ImageHDU()
empty_hdulist = fits.HDUList([hdup,hdu1,hdu2])
empty_hdulist.writeto('empty.fits', overwrite=True)

# Let's look at the file we made
fits.info('empty.fits')
```

```
Filename: empty.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU       4   ()
  1                1 ImageHDU         5   ()
  2                1 ImageHDU         5   ()
```

2.9.6 fxheader

Please review the *Notes* section above before running any examples in this notebook

Fxheader lists one line of header description per FITS unit. This functionality has been replaced in a convenience function in `astropy`, astropy.io.fits.info. It prints the number, name, version, type, length of header (cards), data shape and format for each extension.

```
# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615003'
Observations.download_products(obsid,productFilename="iczgs3y5q_flt.fits")
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# run fits.info
fits.info('./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits')
```

```
Filename: ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     265   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
```

(continues on next page)

(continued from previous page)

```
  3  DQ            1 ImageHDU        43   (1014, 1014)   int16
  4  SAMP          1 ImageHDU        37   (1014, 1014)   int16
  5  TIME          1 ImageHDU        37   (1014, 1014)   float32
  6  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
```

2.9.7 fxplf

Please review the *Notes* section above before running any examples in this notebook

fxplf is used to convert a pixel list file into a BINTABLE extension. We show a simple example below, see the Astropy unified read/write documentation for more details.

```
# Astronomy Specific Imports
from astropy.io import fits
from astropy.table import Table
```

```
# Define input and output files
infile = '../data/table3.txt'
outfile = 'table3.fits'

# read txt, write to fits
t = Table.read(infile, format='ascii')
print(t)
t.write(outfile, overwrite=True)
```

```
col1 col2
---- ----
 200   45
  34  222
   3    4
 100  200
   8   88
  23  123
```

2.9.8 Not Replacing

- funpack - Uncompress FITS file, can be done by opening and resaving file with astropy.io.fits
- fxconvert - Convert between IRAF image types. See **images.imutil.imcopy**
- fgread - Read a MEF file with FOREIGN extensions. Deprecated.
- fgwrite - Create a MEF file with FOREIGN extensions. Deprecated.

2.10 stsdas.toolbox.headers

This package provides utilities for comparing and editing image headers.

2.10.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Many of the headers tasks can be replaced with utilities in `astropy`, as seen below.

Contents:

2.10.2 hdiff

Please review the *Notes* **section above before running any examples in this notebook**

The hdiff task will take two FITS headers and report the differences between them. This functionality has been replaced and improved upon in `astropy` with the `astropy.io.fits.Differs` class, which can be easily called with the printdiff convenience function. For more details on a more advanced differ result using `astropy.io.fits.Differs` directly, see the API doc.

```
# Astronomy Specific Imports
from astropy.io import fits
from astropy.io.fits import printdiff
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
file1 = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
file2 = './mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits'

# printdiff example ignoring HISTORY and COMMENT cards, and only extension 0
printdiff(file1, file2, ext=0, ignore_keywords=('HISTORY', 'COMMENT'))
```

```
Headers contain differences:
  Headers have different number of cards:
   a: 225
```

(continues on next page)

(continued from previous page)

```
 b: 242
Extra keyword 'ANG_SIDE' in a: 0.0
Extra keyword 'BIACFILE' in a: 'N/A'
Extra keyword 'CCDOFSAB' in a: 190
Extra keyword 'CCDOFSCD' in a: 190
Extra keyword 'CSMID'  in a: 'IR'
Extra keyword 'DWELL_LN' in a: 0
Extra keyword 'DWELL_TM' in a: 0.0
Extra keyword 'FILTER' in a: 'F140W'
Extra keyword 'MYKEY1' in a: 5
Extra keyword 'NLINCORR' in a: 'COMPLETE'
Extra keyword 'NLINFILE' in a: 'iref$u1k1727mi_lin.fits'
Extra keyword 'NO_LINES' in a: 0
Extra keyword 'NSAMP'  in a: 14
Extra keyword 'PHOTBW' in a: 1132.39
Extra keyword 'PHOTFLAM' in a: 1.4737148e-20
Extra keyword 'PHOTFNU' in a: 9.5291135e-08
Extra keyword 'PHOTMODE' in a: 'WFC3 IR F140W'
Extra keyword 'PHOTPLAM' in a: 13922.907
Extra keyword 'PHOTZPT' in a: -21.1
Extra keyword 'SAACRMAP' in a: 'N/A'
Extra keyword 'SAA_DARK' in a: 'N/A'
Extra keyword 'SAA_EXIT' in a: '2016.015:05:44:00'
Extra keyword 'SAA_TIME' in a: 3808
Extra keyword 'SAMPZERO' in a: 2.911756
Extra keyword 'SAMP_SEQ' in a: 'SPARS50'
Extra keyword 'SCAN_ANG' in a: 0.0
Extra keyword 'SCAN_COR' in a: 'C'
Extra keyword 'SCAN_LEN' in a: 0.0
Extra keyword 'SCAN_RAT' in a: 0.0
Extra keyword 'SCAN_TYP' in a: 'N'
Extra keyword 'SCAN_WID' in a: 0.0
Extra keyword 'SUBTYPE' in a: 'FULLIMAG'
Extra keyword 'UNITCORR' in a: 'COMPLETE'
Extra keyword 'ZOFFCORR' in a: 'COMPLETE'
Extra keyword 'ZSIGCORR' in a: 'COMPLETE'
Extra keyword 'ATODCORR' in b: 'OMIT'
Extra keyword 'BIASCORR' in b: 'COMPLETE'
Extra keyword 'CCDOFSTA' in b: 1
Extra keyword 'CCDOFSTB' in b: 1
Extra keyword 'CCDOFSTC' in b: 1
Extra keyword 'CCDOFSTD' in b: 1
Extra keyword 'CFLTFILE' in b: 'N/A'
Extra keyword 'CRSPLIT' in b: 1
Extra keyword 'CTEDATE0' in b: 52334.86
Extra keyword 'CTEDATE1' in b: 57710.4460102
Extra keyword 'CTEDIR' in b: 'NONE'
Extra keyword 'CTEIMAGE' in b: 'NONE'
Extra keyword 'CTE_NAME' in b: 'PixelCTE 2017'
Extra keyword 'CTE_VER' in b: '1.2'
Extra keyword 'DARKTIME' in b: 581.247202
Extra keyword 'EXPSCORR' in b: 'COMPLETE'
Extra keyword 'FILTER1' in b: 'CLEAR1L'
Extra keyword 'FILTER2' in b: 'F814W'
Extra keyword 'FIXROCR' in b: 1
Extra keyword 'FLASHCUR' in b: 'OFF'
Extra keyword 'FLASHDUR' in b: 0.0
```

(continues on next page)

(continued from previous page)

```
 Extra keyword 'FLASHSTA' in b: 'NOT PERFORMED'
 Extra keyword 'FLSHCORR' in b: 'OMIT'
 Extra keyword 'FW1ERROR' in b: False
 Extra keyword 'FW1OFFST' in b: 0
 Extra keyword 'FW2ERROR' in b: False
 Extra keyword 'FW2OFFST' in b: 0
 Extra keyword 'FWSERROR' in b: False
 Extra keyword 'FWSOFFST' in b: 0
 Extra keyword 'JWROTYPE' in b: 'DS_int'
 Extra keyword 'LRFWAVE' in b: 0.0
 Extra keyword 'MLINTAB' in b: 'N/A'
 Extra keyword 'PCTECORR' in b: 'COMPLETE'
 Extra keyword 'PCTEFRAC' in b: 0.9937865427707
 Extra keyword 'PCTENFOR' in b: 5
 Extra keyword 'PCTENPAR' in b: 7
 Extra keyword 'PCTERNOI' in b: 4.3
 Extra keyword 'PCTETLEN' in b: 60
 Extra keyword 'PCTETRSH' in b: -10.0
 Extra keyword 'PHOTTAB' in b: 'N/A'
 Extra keyword 'SHADCORR' in b: 'OMIT'
 Extra keyword 'SHADFILE' in b: 'N/A'
 Extra keyword 'SHUTRPOS' in b: 'A'
 Extra keyword 'SINKCORR' in b: 'COMPLETE'
 Extra keyword 'SPOTTAB' in b: 'N/A'
 Extra keyword 'STATFLAG' in b: False
 Extra keyword 'WRTERR' in b: True
 Inconsistent duplicates of keyword ''      :
  Occurs 19 time(s) in a, 17 times in (b)
 Keyword         [8] has different values:
    a>       / INSTRUMENT CONFIGURATION INFORMATION
     ?                                 ------------
    b>       / SCIENCE INSTRUMENT CONFIGURATION
     ?        ++++++++
 Keyword         [9] has different values:
    a>       / POST-SAA DARK KEYWORDS
    b>       / CALIBRATION SWITCHES: PERFORM, OMIT, COMPLETE
 Keyword         [10] has different values:
    a>       / SCAN KEYWORDS
    b>       / CALIBRATION REFERENCE FILES
 Keyword         [11] has different values:
    a>       / CALIBRATION SWITCHES: PERFORM, OMIT, COMPLETE, SKIPPED
    b>       / COSMIC RAY REJECTION ALGORITHM PARAMETERS
 Keyword         [12] has different values:
    a>       / CALIBRATION REFERENCE FILES
    b>       / OTFR KEYWORDS
 Keyword         [13] has different values:
    a>       / COSMIC RAY REJECTION ALGORITHM PARAMETERS
    b>       / PATTERN KEYWORDS
 Keyword         [14] has different values:
    a>       / PHOTOMETRY KEYWORDS
    b>       / POST FLASH  PARAMETERS
 Keyword         [15] has different values:
    a>       / OTFR KEYWORDS
    b>       / ENGINEERING PARAMETERS
 Keyword         [16] has different values:
    a>       / PATTERN KEYWORDS
    b>       / CALIBRATED ENGINEERING PARAMETERS
```

(continues on next page)

(continued from previous page)

```
Keyword         [17] has different values:
   a>       / ENGINEERING PARAMETERS
   b>       / ASSOCIATION KEYWORDS
Keyword APERTURE has different values:
   a> IR-FIX
   b> WFCENTER
Keyword ASN_ID   has different values:
   a> NONE
   b> JCZGX1020
Keyword ASN_MTYP has different values:
   b> EXP-DTH
Keyword ASN_TAB  has different values:
   a> NONE
   b> jczgx1020_asn.fits
Keyword ATODGNA  has different values:
   a> 2.3399999
   b> 2.02
Keyword ATODGNB  has different values:
   a> 2.3699999
   b> 1.886
Keyword ATODGNC  has different values:
   a> 2.3099999
   b> 2.017
Keyword ATODGND  has different values:
   a> 2.3800001
   b> 2.0109999
Keyword ATODTAB  has different comments:
   b> analog to digital correction file
Keyword BIASFILE has different values:
   a> N/A
   b> jref$1541940gj_bia.fits
Keyword BIASFILE has different comments:
   b> bias image file name
Keyword BIASLEVA has different values:
   a> 0.0
   b> 4221.167
Keyword BIASLEVB has different values:
   a> 0.0
   b> 4029.7478
Keyword BIASLEVC has different values:
   a> 0.0
   b> 4441.6987
Keyword BIASLEVD has different values:
   a> 0.0
   b> 4631.4839
Keyword BITPIX   has different comments:
   b> number of bits per data pixel
Keyword BLEVCORR has different comments:
   a> subtract bias level computed from ref pixels
    ?                                    ^^^^ ^^^^
   b> subtract bias level computed from overscan img
    ?                                   +++ ^^^^^ ^^
Keyword BPIXTAB  has different values:
   a> iref$y711520di_bpx.fits
   b> jref$t3n1116nj_bpx.fits
Keyword CAL_VER  has different values:
   a> 3.3(28-Jan-2016)
```

(continues on next page)

(continued from previous page)

```
   b> 9.2.0 (01-Jun-2017)
Keyword CAL_VER  has different comments:
   a> CALWF3 code version
    ?    ^^^
   b> CALACS code version
    ?    ^^^
Keyword CCDGAIN  has different values:
   a> 2.5
   b> 2.0
Keyword CCDTAB   has different values:
   a> iref$t2c16200i_ccd.fits
   b> jref$xa81715gj_ccd.fits
Keyword CCDTAB   has different comments:
   a> detector calibration parameters
    ? ^^^^^^^^
   b> CCD calibration parameters
    ? ^^^
Keyword CRCORR   has different values:
   a> COMPLETE
   b> OMIT
Keyword CRCORR   has different comments:
   a> identify cosmic ray hits
   b> combine observations to reject cosmic rays
Keyword CRDS_CTX has different values:
   a> hst_0478.pmap
    ?      ^^^
   b> hst_0592.pmap
    ?      ^^^
Keyword CRDS_VER has different values:
   a> 7.0.1, opus_2016.1-universal, af27872
   b> 7.1.5, 7.1.5, 3548bc1
Keyword CRREJTAB has different values:
   a> iref$u6a1748ri_crr.fits
   b> N/A
Keyword CSYS_VER has different values:
   a> hstdp-2016.1
    ?          ^ ^
   b> hstdp-2017.3
    ?          ^ ^
Keyword D2IMFILE has different values:
   a> N/A
   b> jref$02c1450oj_d2i.fits
Keyword D2IMFILE has different comments:
   b> Column Correction Reference File
Keyword DARKFILE has different values:
   a> iref$xag19296i_drk.fits
   b> jref$19k1602ij_drk.fits
Keyword DATE     has different values:
   a> 2016-09-21
   b> 2017-12-03
Keyword DATE-OBS has different values:
   a> 2016-01-15
    ?       -  ^
   b> 2016-10-16
    ?      +   ^
Keyword DEC_TARG has different values:
   a> 48.92264646942
```

(continues on next page)

(continued from previous page)

```
    b> 65.84194444444
 Keyword DETECTOR has different values:
    a> IR
    b> WFC
 Keyword DETECTOR has different comments:
    a> detector in use: UVIS or IR
    b> detector in use: WFC, HRC, or SBC
 Keyword DGEOFILE has different values:
    a> N/A
    b> jref$qbu16429j_dxy.fits
 Keyword DISTNAME has different values:
    a> iczgs3ygq_w3m18525i-NOMODEL-NOMODEL
    b> jczgx1ppq_11d1433lj-02c1450rj-02c1450oj
 Keyword DRIZCORR has different values:
    a> COMPLETE
    b> PERFORM
 Keyword DRKCFILE has different values:
    a> N/A
    b> jref$19k15450j_dkc.fits
 Keyword DRKCFILE has different comments:
    b> De-trailed Dark Reference File
 Keyword EXPEND   has different values:
    a> 57402.29030181
    b> 57677.05173856
 Keyword EXPSTART has different values:
    a> 57402.28332292
    b> 57677.04503644
 Keyword EXPTIME  has different values:
    a> 602.937317
    b> 578.0
 Keyword FILENAME has different values:
    a> iczgs3ygq_flt.fits
    b> jczgx1ppq_flc.fits
 Keyword FLSHFILE has different comments:
    b> post flash correction file name
 Keyword IDCTAB   has different values:
    a> iref$w3m18525i_idc.fits
    b> jref$11d1433lj_idc.fits
 Keyword IMPHTTAB has different values:
    a> iref$wbj1825ri_imp.fits
    b> jref$08b18470j_imp.fits
 Keyword INSTRUME has different values:
    a> WFC3
    b> ACS
 Keyword LINENUM  has different values:
    a> S3.008
    b> X1.009
 Keyword MDRIZTAB has different values:
    a> iref$ubi1853pi_mdz.fits
    b> jref$16r12191j_mdz.fits
 Keyword MOONANGL has different values:
    a> 57.153374
    b> 92.141869
 Keyword NAXIS    has different comments:
    b> number of data axes
 Keyword NEXTEND  has different values:
    a> 6
```

(continues on next page)

(continued from previous page)

```
   b> 15
Keyword NPOLFILE has different values:
   a> N/A
   b> jref$02c1450rj_npl.fits
Keyword NPOLFILE has different comments:
   b> Non-polynomial Offsets Reference File
Keyword OBSMODE  has different values:
   a> MULTIACCUM
   b> ACCUM
Keyword OPUS_VER has different values:
   a> HSTDP 2016_1a
    ?          ^ ^^
   b> HSTDP 2017_3
    ?          ^ ^
Keyword ORIGIN   has different comments:
   b> FITS file originator
Keyword OSCNTAB  has different values:
   a> iref$q911321mi_osc.fits
   b> jref$17717071j_osc.fits
Keyword OSCNTAB  has different comments:
   a> detector overscan table
   b> CCD overscan table
Keyword PA_V3    has different values:
   a> 282.776093
   b> 88.003448
Keyword PCTETAB  has different values:
   a> N/A
   b> jref$19i16323j_cte.fits
Keyword PCTETAB  has different comments:
   b> CTE Correction Table
Keyword PFLTFILE has different values:
   a> iref$uc721143i_pfl.fits
   b> jref$qb12257pj_pfl.fits
Keyword PROCTIME has different values:
   a> 57652.2953588
   b> 58090.39077546
Keyword PROPAPER has different values:
   b> WFCENTER
Keyword PYWCSVER has different values:
   a> 1.2.1
   b> 1.3.3
Keyword RA_TARG  has different values:
   a> 36.85374208875
   b> 127.7389583333
Keyword READNSEA has different values:
   a> 20.200001
   b> 4.3499999
Keyword READNSEB has different values:
   a> 19.799999
   b> 3.75
Keyword READNSEC has different values:
   a> 19.9
   b> 4.0500002
Keyword READNSED has different values:
   a> 20.1
   b> 5.0500002
Keyword ROOTNAME has different values:
```

(continues on next page)

(continued from previous page)

```
    a> iczgs3ygq
    b> jczgx1ppq
  Keyword RPTCORR  has different comments:
    a> combine individual repeat observations
     ? ^^^^^^^
    b> add individual repeat observations
     ? ^^^
  Keyword SIPNAME  has different values:
    a> iczgs3ygq_w3m18525i
    b> jczgx1ppq_11d1433lj
  Keyword SNKCFILE has different values:
    a> N/A
    b> jref$16q1417cj_snk.fits
  Keyword SNKCFILE has different comments:
    b> Map of sink pixels
  Keyword SUNANGLE has different values:
    a> 112.720184
    b> 91.557938
  Keyword SUN_ALT  has different values:
    a> 3.227515
    b> 54.863163
  Keyword TARGNAME has different values:
    a> ANY
    b> ACO-665
  Keyword TIME-OBS has different values:
    a> 06:47:59
    b> 01:04:51
  Keyword UPWCSVER has different values:
    a> 1.2.3.dev
    b> 1.3.2
```

2.10.3 stfhistory

Please review the *Notes* section above before running any examples in this notebook

The stfhistory task will read history information from a text file and add it to an image header. Here we will show how to do this with a FITS file using Python's built in i/o functionality and the `astropy.io.fits` package.

```
# Standard Imports
import shutil

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004663553'
Observations.download_products(obsid,productFilename="jczgx1ppq_flc.fits")
```

```
INFO: Found cached file ./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits with expected␣
↪size 167964480. [astroquery.query]
```

```
# open our text file and fits file objects, we're going to make a copy of a fits file,
↪ and edit the copy
my_file = open('../data/history_info.txt', 'r')
shutil.copyfile('./mastDownload/HST/JCZGX1PPQ/jczgx1ppq_flc.fits','stfhist_copy.fits')
test_data = fits.open('stfhist_copy.fits', mode='update')

# loop through lines in text file and write to fits file
# here we add the HISTORY lines to the zeroth header
for line in my_file:
    test_data[0].header.add_history(line.strip('\n'))

# make sure to close your files after the edits are done
test_data.close()
my_file.close()
```

2.10.4 Not Replacing

- eheader - Interactively edit an image header. Deprecated.
- groupmod - GEIS header editing. Deprecated, for FITS header editing see **images.imutil.hedit**
- hcheck - see **images.imutil.hselect**
- iminfo - see **images.imutil.imheader**
- upreffile - Update calibration reference files names in image headers. See crds package

2.11 tables.fitsio

The tables.fitsio package contains IO utilities for FITS and GEIS images.

2.11.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Many of the tasks in the this package are no longer in common usage and are not covered here. If there is a task you would like to request please contact the STAK team.

Contents:

- *catfits*
- *stwfits*

2.11.2 catfits

Please review the *Notes* section above before running any examples in this notebook

The catfits task was used to quickly produce a catalog of fits headers from a file list. In the below example we provide the summary catalog provided by `astropy.io.fits`.

```
# Standard Imports
import glob

# Astronomy Specific Imports
from astropy.io import fits
```

```
# Change these values to your desired data files, glob will capture all wildcard␣
↪matches
test_data = glob.glob('../data/*.fits')

for filename in test_data:
    fits.info(filename)
```

```
Filename: ../data/imstack_out.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  SCI           1 PrimaryHDU     199   (4096, 2048, 2)   float32
Filename: ../data/jczgx1ppq_flc.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     279   ()
  1  SCI           1 ImageHDU       200   (4096, 2048)   float32
  2  ERR           1 ImageHDU        56   (4096, 2048)   float32
  3  DQ            1 ImageHDU        48   (4096, 2048)   int16
  4  SCI           2 ImageHDU       198   (4096, 2048)   float32
  5  ERR           2 ImageHDU        56   (4096, 2048)   float32
  6  DQ            2 ImageHDU        48   (4096, 2048)   int16
  7  D2IMARR       1 ImageHDU        15   (64, 32)   float32
  8  D2IMARR       2 ImageHDU        15   (64, 32)   float32
  9  D2IMARR       3 ImageHDU        15   (64, 32)   float32
 10  D2IMARR       4 ImageHDU        15   (64, 32)   float32
 11  WCSDVARR      1 ImageHDU        15   (64, 32)   float32
 12  WCSDVARR      2 ImageHDU        15   (64, 32)   float32
 13  WCSDVARR      3 ImageHDU        15   (64, 32)   float32
 14  WCSDVARR      4 ImageHDU        15   (64, 32)   float32
 15  WCSCORR       1 BinTableHDU     59   14R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
Filename: ../data/jczgx1ptq_flc.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     279   ()
  1  SCI           1 ImageHDU       200   (4096, 2048)   float32
  2  ERR           1 ImageHDU        56   (4096, 2048)   float32
  3  DQ            1 ImageHDU        48   (4096, 2048)   int16
  4  SCI           2 ImageHDU       198   (4096, 2048)   float32
  5  ERR           2 ImageHDU        56   (4096, 2048)   float32
  6  DQ            2 ImageHDU        48   (4096, 2048)   int16
  7  D2IMARR       1 ImageHDU        15   (64, 32)   float32
  8  D2IMARR       2 ImageHDU        15   (64, 32)   float32
  9  D2IMARR       3 ImageHDU        15   (64, 32)   float32
 10  D2IMARR       4 ImageHDU        15   (64, 32)   float32
 11  WCSDVARR      1 ImageHDU        15   (64, 32)   float32
 12  WCSDVARR      2 ImageHDU        15   (64, 32)   float32
```

(continues on next page)

(continued from previous page)

```
 13  WCSDVARR      3 ImageHDU        15   (64, 32)   float32
 14  WCSDVARR      4 ImageHDU        15   (64, 32)   float32
 15  WCSCORR       1 BinTableHDU     59   14R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
Filename: ../data/jczgx1q1q_flc.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     279   ()
  1  SCI           1 ImageHDU       200   (4096, 2048)   float32
  2  ERR           1 ImageHDU        56   (4096, 2048)   float32
  3  DQ            1 ImageHDU        48   (4096, 2048)   int16
  4  SCI           2 ImageHDU       198   (4096, 2048)   float32
  5  ERR           2 ImageHDU        56   (4096, 2048)   float32
  6  DQ            2 ImageHDU        48   (4096, 2048)   int16
  7  D2IMARR       1 ImageHDU        15   (64, 32)   float32
  8  D2IMARR       2 ImageHDU        15   (64, 32)   float32
  9  D2IMARR       3 ImageHDU        15   (64, 32)   float32
 10  D2IMARR       4 ImageHDU        15   (64, 32)   float32
 11  WCSDVARR      1 ImageHDU        15   (64, 32)   float32
 12  WCSDVARR      2 ImageHDU        15   (64, 32)   float32
 13  WCSDVARR      3 ImageHDU        15   (64, 32)   float32
 14  WCSDVARR      4 ImageHDU        15   (64, 32)   float32
 15  WCSCORR       1 BinTableHDU     59   14R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
Filename: ../data/nnicqr34r1q_blv_tmp.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     314   ()
  1  SCI           1 ImageHDU        89   (4096, 2051)   float32
  2  ERR           1 ImageHDU        45   (4096, 2051)   float32
  3  DQ            1 ImageHDU        71   (4096, 2051)   int16
  4  SCI           2 ImageHDU        89   (4096, 2051)   float32
  5  ERR           2 ImageHDU        45   (4096, 2051)   float32
  6  DQ            2 ImageHDU        71   (4096, 2051)   int16
Filename: ../data/nnicqr34rgq_blv_tmp.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     314   ()
  1  SCI           1 ImageHDU        89   (4096, 2051)   float32
  2  ERR           1 ImageHDU        45   (4096, 2051)   float32
  3  DQ            1 ImageHDU        71   (4096, 2051)   int16
  4  SCI           2 ImageHDU        89   (4096, 2051)   float32
  5  ERR           2 ImageHDU        45   (4096, 2051)   float32
  6  DQ            2 ImageHDU        71   (4096, 2051)   int16
Filename: ../data/nnicqr34rvq_blv_tmp.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     314   ()
  1  SCI           1 ImageHDU        89   (4096, 2051)   float32
  2  ERR           1 ImageHDU        45   (4096, 2051)   float32
  3  DQ            1 ImageHDU        71   (4096, 2051)   int16
  4  SCI           2 ImageHDU        89   (4096, 2051)   float32
  5  ERR           2 ImageHDU        45   (4096, 2051)   float32
  6  DQ            2 ImageHDU        71   (4096, 2051)   int16
Filename: ../data/stfhist.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     266   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
  3  DQ            1 ImageHDU        43   (1014, 1014)   int16
  4  SAMP          1 ImageHDU        37   (1014, 1014)   int16
```

(continues on next page)

(continued from previous page)

```
  5  TIME          1 ImageHDU        37   (1014, 1014)   float32
  6  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
Filename: ../data/wfc3data_flt.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     265   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
  3  DQ            1 ImageHDU        43   (1014, 1014)   int16
  4  SAMP          1 ImageHDU        37   (1014, 1014)   int16
  5  TIME          1 ImageHDU        37   (1014, 1014)   float32
  6  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
```

2.11.3 stwfits

Please review the *Notes* section above before running any examples in this notebook

stwfits is used to translate a GEIS (Generic Edited Information Set), STSDAS tables, or ascii file to an standard FITS(Flexible Image Transport System) format. Here we will cover how to convert a GEIS file to a FITS files using the `stsci.tools.readgeis` function. There are two ways to use this function, through the command line, or through a Python session or script. For instructions on running this task on the command line see the `stsci.tools` Conversion Utilities documentation. Below we show an example of running this task in a python session. You may or may not need to byteswap your image data depending on which system it was originally written on.

Below we show an example with a local file. **This example will not run unless the filename is replaced with one of your local files.**

```
# Standard Imports
import glob

# Astronomy Specific Imports
from stsci.tools import readgeis
```

```
filename = "x31g0108t.c0h"
hdulist = readgeis.readgeis(filename)
hdulist[1].data = hdulist[1].data.byteswap()
del hdulist[1].header['CD1_1']
del hdulist[1].header['CD2_2']
hdulist.writeto('stwfits_out.fits', overwrite = True)
```

```
===================================
= WARNING:                        =
=  Input image:                   =
../data/x31g0108t.c0h[1]
=  had floating point data values =
=  of NaN and/or Inf.             =
===================================
===================================
=  This file may have been        =
=  written out on a platform      =
=  with a different byte-order.   =
=                                 =
=  Please verify that the values  =
```

(continues on next page)

(continued from previous page)

```
=  are correct or apply the       =
=  '.byteswap()' method.          =
==================================
```

2.11.4 Not Replacing

- fits_example - used to provide more documentation for stwfits and strfits
- fitscopy - used to produce a copy of a fits file, producing a copy of a fits file is straightforward in Python and the command line using exsisting libraries
- geis - used to provide a description of GEIS file format
- gftoxdim - GEIS conversion, no longer in common usage
- strfits - converts FITS files to GEIS or STSDAS tables, no longer in common usage
- xdimtogf - convert single group GEIS to multigroup GEIS, no longer in common usage

2.12 tables.ttools

Ttools is an IRAF library used to build and manipulate STSDAS tables.

2.12.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Many of the table tools in this package are easily accessible with the Astropy Table object. Here we will show the `Astropy Table` equivalent of the ttools tasks. You can also find a lot of useful information about tables and more advanced read and write options on the Unified I/O Astropy documentation page.

Below we import the example table we will be using. **Before using some of the examples in this notebook you will need to setup the test table by running the code cell below**

Contents:

- *tiimage-titable-tximage-txtable*
- *tinfo-tlcol-tprint*
- *tintegrate*
- *tjoin*
- *tlinear*
- *tmatch*
- *tmerge*
- *tselect-tproject-tquery*
- *tsort*
- *tstat*

```
#Here we import the example table we will be using from a text file:
from astropy.table import Table

filename = "../data/table2.txt"
ex_table = Table.read(filename, format='ascii')
ex_table
```

2.12.2 imtab-tabim

Please review the *Notes* section above before running any examples in this notebook

Imtab can be used to copy an image to a table column. We can accomplish this by first flattening the array (2D down to 1D), then putting it into a table. For more details see the Table construction documentation. Tabim is used to copy a column back to a table, as show below.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.table import Table
```

```
# Create test array and flatten
image_array=np.random.rand(6,6)
image_array=image_array.flatten()

# Put into table, to make it a column we need the outside []
t = Table([image_array])
print(t)

# Now to re-extract the array we have to grab the
# data and unflatten it. The column was given the
# default name of col0 by Table
extract_array = t['col0'].data.reshape((6,6))
print(extract_array)
```

```
      col0
---------------
  0.73498113873
 0.601683040128
```

(continues on next page)

(continued from previous page)

```
 0.858365279296
 0.183850195764
 0.372479856903
 0.531179886849
 0.497759057246
  0.24850881731
 0.433906702747
0.0199450763848
            ...
0.0908400575378
 0.448676070596
 0.275824527206
 0.276164794467
 0.193654333786
 0.830174255037
 0.581290249067
 0.754640533974
 0.651459214252
 0.435245983443
  0.75900952991
Length = 36 rows
[[ 0.73498114  0.60168304  0.85836528  0.1838502   0.37247986  0.53117989]
 [ 0.49775906  0.24850882  0.4339067   0.01994508  0.4251196   0.53538164]
 [ 0.8670757   0.38572518  0.39294164  0.34951696  0.53854753  0.8362706 ]
 [ 0.68752468  0.4442957   0.33628146  0.75661578  0.87014016  0.88223051]
 [ 0.3725361   0.09084006  0.44867607  0.27582453  0.27616479  0.19365433]
 [ 0.83017426  0.58129025  0.75464053  0.65145921  0.43524598  0.75900953]]
```

2.12.3 partab

Please review the *Notes* section above before running any examples in this notebook

Partab is used to transfer an IRAF parameter to a table element. Below we show the `Astropy Table` equivalent using indexing. See the Modifying Table documentation for more details.

```
# Astronomy Specific Imports
from astropy.table import Table
```

```
ex_table['fwhm'][4]=4.5
ex_table
```

2.12.4 tabpar

Please review the *Notes* section above before running any examples in this notebook

The tabpar task takes a header keyword and moves it to an IRAF parameter. Extracting values from an astropy table is straightfoward with indexing. Keep in mind the indexing is zero based. When an FITS file is read into a table, the header information is saved in the metadata as an `Ordered Dictionary`. Below we show you how to pull values from the table data, and metadata.

```
# Astronomy Specific Imports
from astropy.table import Table
```

```
# Pulling a column out of a table
column=ex_table['sname']
print(column)

# Pulling a value out of a table
entry=ex_table['radius'][2]
print('\n')
print(entry)
```

```
sname
-----
star1
star2
star3
star4
star5


2
```

```
# Pulling values out of the metadata
fits_file = '../data/08b18470j_imp.fits'
fits_table = Table.read(fits_file, hdu=2)
print(fits_table.meta)
print(fits_table.meta['EXTNAME'])
```

```
OrderedDict([('EXTNAME', 'PHOTPLAM'), ('EXTVER', 1)])
PHOTPLAM
```

2.12.5 taextract-tainsert

Please review the *Notes* section above before running any examples in this notebook

Taextract and tainsert are used to copy scalar columns to array entries, and vice versa. We will show how to store an array in an `Astropy Table` from a `list` of scalars.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.table import Table
```

```
scalar_list = [4,5,6,7,8,9]

# Change to numpy array
in_arr = np.array(scalar_list)

# Store in table
t = Table([in_arr])
t.pprint()

print("\n")

# Now extract array back to scalar list, flatten will take out the extra dimension
```

(continues on next page)

(continued from previous page)

```
out_arr = t['col0'].data
print(out_arr)
```

```
col0
----
   4
   5
   6
   7
   8
   9


[4 5 6 7 8 9]
```

2.12.6 tcalc

Please review the *Notes* section above before running any examples in this notebook

Tcalc is used to perform arithmetic operations on table columns. This can be done automatically with any compatible data types. A new `Column` object will be returned, which you can add back into the original Table, or a new Table as desired. See the Table modification documentation for more details.

```
# Astronomy Specific Imports
from astropy.table import Table
```

```
out_column = ex_table['radius'] + ex_table['fwhm']
out_column.name = 'radfw'
print(out_column)
```

```
radfw
-----
 16.5
 12.1
  2.5
 1.75
 24.5
```

2.12.7 tchcol

Please review the *Notes* section above before running any examples in this notebook

tchcol is used to change the column name, format or units. This can be done easily with `Astropy Tables`, and the Astropy Units module.

```
# Astronomy Specific Imports
from astropy.table import Table
import astropy.units as u
import numpy as np
```

```
# Set filename, read in file
filename = "../data/table2.txt"
```

(continues on next page)

(continued from previous page)

```
ed_table = Table.read(filename, format='ascii')

# To get table info
print(ed_table.info)

# To add/update units
ed_table['radius'].unit = u.astrophys.pix
print(ed_table.info)

# To change column name
ed_table['radius'].name='radius(pix)'
print(ed_table.info)

# To change dtype
ed_table['radius(pix)'] = ed_table['radius(pix)'].astype(float)
print(ed_table.info)

print(ed_table)
```

```
<Table length=5>
 name   dtype
------ -------
 sname    str5
radius   int64
  fwhm float64

<Table length=5>
 name   dtype  unit
------ ------- ----
 sname    str5
radius   int64  pix
  fwhm float64

<Table length=5>
    name     dtype  unit
---------- ------- ----
      sname    str5
radius(pix)   int64  pix
       fwhm float64

<Table length=5>
    name     dtype  unit
---------- ------- ----
      sname    str5
radius(pix) float64  pix
       fwhm float64

sname radius(pix) fwhm
          pix
----- ---------- ----
star1       10.0  6.5
star2        7.0  5.1
star3        2.0  0.5
star4        1.0 0.75
star5       20.0 13.0
```

2.12.8 tcopy-tdump

Please review the *Notes* section above before running any examples in this notebook

Tcopy is used to copy tables, and can save a table to ASCII or FITS format. Similarly, tdump is used to save a table to an ASCII file. We will show both save methods and a copy below. For more details see the unified read/write documentation. For more details on `Table` object copying see the copy versus reference doc section.

Please be aware that there are many possible ASCII write formats provided by Astropy, listed here. In this example we use the default basic formatting.

```
# Astronomy Specific Imports
from astropy.table import Table
```

```
# Make a copy of our example table
tab_copy = ex_table.copy()

# Save as ASCII
outfile = 'copy_table.txt'
tab_copy.write(outfile, format='ascii', overwrite=True)

# Same method call to write to FITS
outfits = 'copy_table.fits'
tab_copy.write(outfits, overwrite=True)
```

2.12.9 tdiffer

Tdiffer is used to create an output table that is the difference of two tables. Astropy has this functionality in the setdiff function.

```
from astropy.table import Table
from astropy.table import setdiff
```

```
# Setup sample tables
t1 = Table({'a': [1, 4, 9], 'b': ['c', 'd', 'f']}, names=('a', 'b'))
t2 = Table({'a': [1, 5, 9], 'b': ['c', 'b', 'f']}, names=('a', 'b'))

print("table 1: \n{}\n".format(t1))
print("table 2: \n{}\n".format(t2))

# Calculate and print the difference between tables
print("table diff t1-t2")
print(setdiff(t1, t2))

# Same, but t2-t1 instead of t1-t2
print("table diff t2-t1")
print(setdiff(t2, t1))
```

```
table 1:
 a   b
--- ---
  1   c
  4   d
  9   f

```

(continues on next page)

(continued from previous page)

```
table 2:
 a   b
--- ---
  1   c
  5   b
  9   f

table diff t1-t2
 a   b
--- ---
  4   d
table diff t2-t1
 a   b
--- ---
  5   b
```

2.12.10 texpand

Please review the *Notes* section above before running any examples in this notebook

Texpand is used to edit and change tables according to a set of user provided rules. This can be done by building a customized loop over the input table. Below we show a simple example, but this can be easily modified to fit the users needs.

```
# Astronomy Specific Imports
from astropy.table import Table
```

```
# Change star1 and star2 to a raidus of 10
# Making a copy of the table for editing
new_table = ex_table.copy()
# Loops over the rows in the table
for row in new_table:
    # here we index the columns with numbers
    if row[0] in ['star1','star3']:
        row[1]= 10
print(new_table)
```

```
sname radius fwhm
----- ------ ----
star1     10  6.5
star2      7  5.1
star3     10  0.5
star4      1 0.75
star5     20  4.5
```

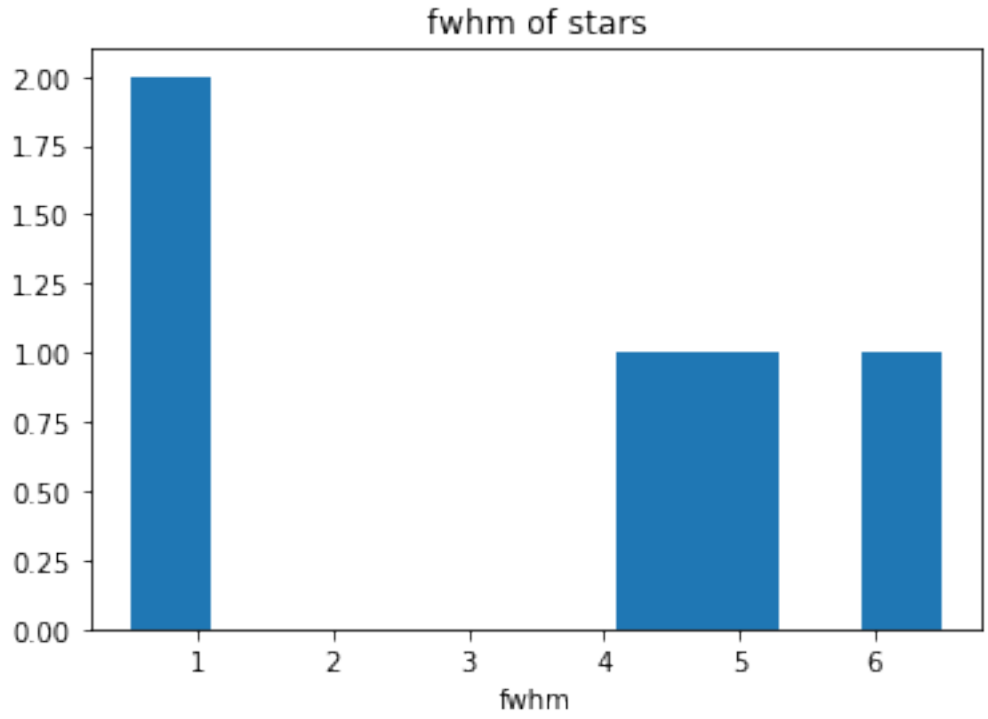
2.12.11 thistogram

Please review the *Notes* section above before running any examples in this notebook

Thistogram makes a histogram from a data column in a table. We can easily accomplish this using the `Astropy Tables` and Matplotlib.pyplot.hist tasks. For this example we will use the default binning. There is also an Astropy histogram and a Numpy histogram available for generating the histogram data.

```
# Astronomy Specific Imports
from astropy.table import Table

# Plotting Imports/Setup
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Using the weight column of our example table
n, bins, patches = plt.hist(ex_table['fwhm'].data)

plt.xlabel('fwhm')
plt.title('fwhm of stars')
plt.show()
```



2.12.12 tiimage-titable-tximage-txtable

Please review the *Notes* section above before running any examples in this notebook

Tiimage, titable, tximage, and txtable are all 3-D table functions. `Astropy Table` objects can store any dimension `numpy` arrays in each element, as long as the columns are consistent. Below we show a short example of storing a 3-D array in an `Astropy` Table. Other table functionality behaves the same for 2-D and 3-D table data.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.table import Table
```

```
# Storing a 2-D arrays in one column of a table
arr1 = np.random.rand(60,90)
arr2 = np.random.rand(60,90)
# To retain the 2-D array as an element in the table, make sure you use two sets of␣
↪square brackets
three_table = Table([[arr1,arr2]],names=('Arrays',))
three_table.pprint()

# To pull out one array element, index column name then row numbers
three_table['Arrays'][1]
```

```
         Arrays [60,90]
--------------------------------
0.892760413585 .. 0.283382986211
0.637760881193 .. 0.363642899902
```

```
array([[ 0.63776088,  0.91520904,  0.02255264, ...,  0.68817791,
         0.53479407,  0.30667641],
       [ 0.97267867,  0.55856732,  0.86993039, ...,  0.91039544,
         0.63862112,  0.58102198],
       [ 0.51181066,  0.85164649,  0.05432316, ...,  0.36084783,
         0.58934112,  0.96374561],
       ...,
       [ 0.83594372,  0.79412333,  0.78455287, ...,  0.88604032,
         0.16606121,  0.1500973 ],
       [ 0.81858617,  0.16964881,  0.00841479, ...,  0.66355838,
         0.95266558,  0.79603504],
       [ 0.81294063,  0.79609841,  0.58490711, ...,  0.3697692 ,
         0.65451337,  0.3636429 ]])
```

2.12.13 tinfo-tlcol-tprint

Please review the *Notes* section above before running any examples in this notebook

Tinfo, tlcol and tprint were all used to display information about the table. Below we show the `Astropy Table` equivalents, including `showtable` which is callable from the terminal.

```
# Astronomy Specific Imports
from astropy.table import Table
```

```
# For tinfo and tlcol
print(ex_table.info)
```

```
<Table length=5>
 name   dtype
------ -------
 sname    str5
radius   int64
  fwhm float64
```

```
# For tprint
ex_table.pprint()
```

```
sname radius fwhm
----- ------ ----
star1     10  6.5
star2      7  5.1
star3      2  0.5
star4      1 0.75
star5     20  4.5
```

```
# To print a specific subset of the table
# Here we pull out the sname and fwhm columns
# and rows 1-3
ex_table['sname','fwhm'][0:3]
```

```
# To print a table outside of a Python interpreter
# Astropy has added the convenience function showtable
!showtable --format ascii ../data/table2.txt
```

```
[0;31msname radius fwhm[0m
[0;31m----- ------ ----[0m
star1     10  6.5
star2      7  5.1
star3      2  0.5
star4      1 0.75
star5     20 13.0
```

```
# Here is the showtable help for more details on usage
!showtable --help
```

```
usage: showtable [-h] [--format FORMAT] [--more] [--info] [--stats]
                 [--max-lines MAX_LINES] [--max-width MAX_WIDTH] [--hide-unit]
                 [--show-dtype] [--delimiter DELIMITER] [--hdu HDU]
                 [--path PATH] [--table-id TABLE_ID]
                 filename [filename ...]

Print tables from ASCII, FITS, HDF5, VOTable file(s). The tables are read with
'astropy.table.Table.read' and are printed with 'astropy.table.Table.pprint'.
The default behavior is to make the table output fit onto a single screen
page. For a long and wide table this will mean cutting out inner rows and
columns. To print all the rows or columns use --max-lines=-1 or max-
width=-1, respectively. The complete list of supported formats can be found
at http://astropy.readthedocs.io/en/latest/io/unified.html#built\unhbox\voidb@
```

```
x\kern\z@\char`\protect\discretionary{\char\defaulthyphenchar}{}{}in\unhbox\
voidb@x\kern\z@\char`\protect\discretionary{\char\defaulthyphenchar}{}{}table-
readers-writers

positional arguments:
  filename              path to one or more files

optional arguments:
  -h, --help            show this help message and exit
  --format FORMAT       input table format, should be specified if it cannot
                        be automatically detected
  --more                use the pager mode from Table.more
  --info                show information about the table columns
  --stats               show statistics about the table columns

pprint arguments:
  --max-lines MAX_LINES
                        maximum number of lines in table output
                        (default=screen length, -1 for no limit)
  --max-width MAX_WIDTH
                        maximum width in table output (default=screen width,
                        -1 for no limit)
  --hide-unit           hide the header row for unit (which is shown only if
                        one or more columns has a unit)
  --show-dtype          include a header row for column dtypes

ASCII arguments:
  --delimiter DELIMITER
                        column delimiter string

FITS arguments:
  --hdu HDU             name of the HDU to show

HDF5 arguments:
  --path PATH           the path from which to read the table

VOTable arguments:
  --table-id TABLE_ID   the table to read in
```

2.12.14 tintegrate

Please review the *Notes* section above before running any examples in this notebook

Tintegrate is used to numerically integrate one column with respect to another. This can be done using the numpy.traz function. As we have shown how to extract an array from a Table in various other tasks in this notebook we will only cover the integration step here.

```
# Standard Imports
import numpy as np

# Astronomy Specific Imports
from astropy.table import Table
```

```
# Setup array, here you would pull from a table
x = [1, 2, 3, 4, 6]
y = [10.5, 12.3, 22.2, 13.3, 7.7]

result = np.trapz(y,x)
print(result)
```

```
67.4
```

2.12.15 tjoin

Please review the *Notes* section above before running any examples in this notebook

Tjoin is used to perform a relational join of two tables. You can do all join types (inner, left, right, and outer) in the Astropy `Tables` package, see join docs here for more details. We take the examples shown here from the Astropy docs.

```
# Astronomy Specific Imports
from astropy.table import Table, join
```

```
# Setup tables
optical = Table.read("""name    obs_date    mag_b  mag_v
                        M31     2012-01-02  17.0   16.0
                        M82     2012-10-29  16.2   15.2
                        M101    2012-10-31  15.1   15.5""", format='ascii')
xray = Table.read("""   name    obs_date    logLx
                        NGC3516 2011-11-11  42.1
                        M31     1999-01-05  43.1
                        M82     2012-10-29  45.0""", format='ascii')
```

```
# Default inner join, default key column to set of columns that are common to both␣
↪tables.
opt_xray = join(optical, xray)
print(opt_xray)
```

```
name  obs_date  mag_b mag_v logLx
---- ---------- ----- ----- -----
 M82 2012-10-29  16.2  15.2  45.0
```

```
# Left join
print(join(optical, xray, join_type='left'))
```

```
name  obs_date  mag_b mag_v logLx
---- ---------- ----- ----- -----
M101 2012-10-31  15.1  15.5    --
 M31 2012-01-02  17.0  16.0    --
 M82 2012-10-29  16.2  15.2  45.0
```

```
# Right join, with only name field as key
print(join(optical, xray, join_type='right', keys='name'))
```

```
  name  obs_date_1 mag_b mag_v obs_date_2 logLx
------- ---------- ----- ----- ---------- -----
```

(continues on next page)

(continued from previous page)

```
    M31 2012-01-02  17.0  16.0 1999-01-05  43.1
    M82 2012-10-29  16.2  15.2 2012-10-29  45.0
NGC3516         --    --    -- 2011-11-11  42.1
```

```
# Outer join
print(join(optical, xray, join_type='outer'))
```

```
  name   obs_date  mag_b mag_v logLx
------- ---------- ----- ----- -----
   M101 2012-10-31  15.1  15.5    --
    M31 1999-01-05    --    --  43.1
    M31 2012-01-02  17.0  16.0    --
    M82 2012-10-29  16.2  15.2  45.0
NGC3516 2011-11-11    --    --  42.1
```

2.12.16 tlinear

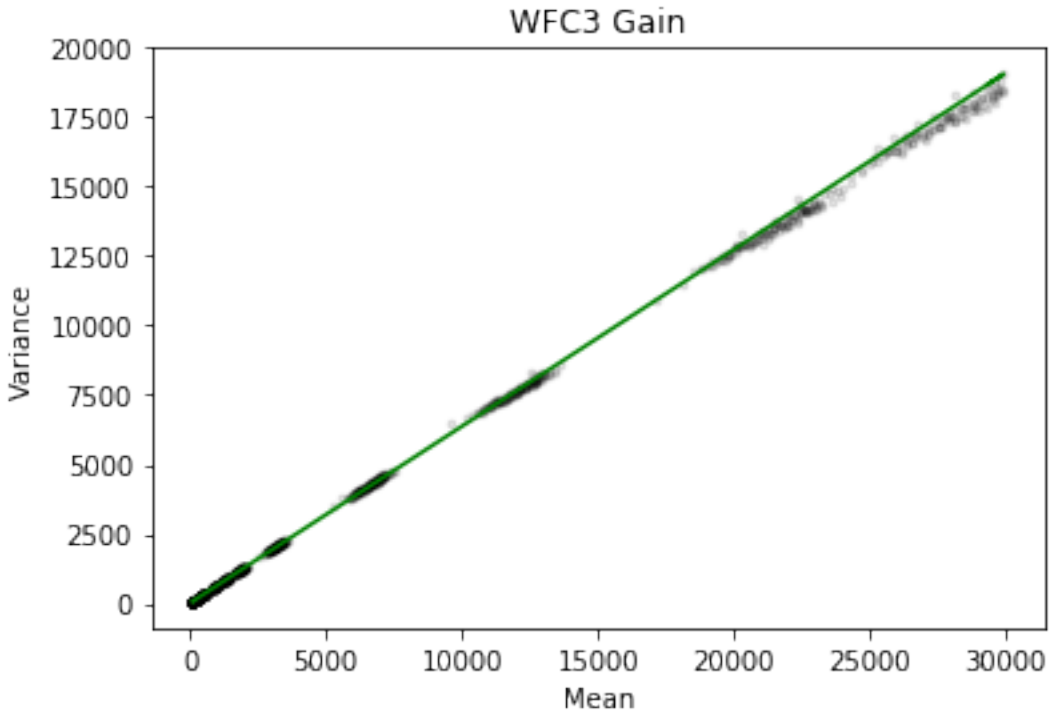
Please review the *Notes* section above before running any examples in this notebook

Tlinear is used to fit a one-dimensional line to a dataset and apply weights with standard deviation. This functionality is contained in the polyfit package and poly1d package of Numpy. Tlinear applies Gaussian weights (inverse STD instead of inverse squared STD) but with this change in mind Polyfit recreates the results of Tlinear to machine precision. We took this example from the WFC3/UVIS Gain Monitor to demonstrate.

```
# All you need is Numpy
import numpy as np

# But Astropy will let us open the file
from astropy.io import ascii

# And Matplotlib will make for better demonstration
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Test dataset taken from WFC3/UVIS Gain Monitor
infile = '../data/WFC3_gain.dat'
data = ascii.read(infile)
variance, mean, std = data['variance'], data['mean'], data['std']

# Fit 1D line to variance vs mean with Gaussian std weights.
fit = np.polyfit(mean, variance, deg=1, w=1/std)
slope, intercept = fit

# Turn the slope and intercept into a functional form and apply it.
fit_line = np.poly1d(fit)
y_line = fit_line(mean)

# Show off our handy work
print('Line fit to the data : y = {}*x + {}'.format(slope, intercept))
plt.scatter(mean, variance, color='black', alpha=.1, s=8)
plt.plot(mean, y_line, color='green')
plt.xlabel('Mean')
plt.ylabel('Variance')
plt.title('WFC3 Gain')
```

```
Line fit to the data : y = 0.6364783735418926*x + 5.130580511662094
```



2.12.17 tmatch

Please review the *Notes* section above before running any examples in this notebook

Tmatch is used to find the closest match between rows in two tables. This functionality is contained in the coordinates package of Astropy. This example is taken from the Coordinates notebook, please see the notebook for more details before expanding this example to suit your needs.

```
# Astronomy Specific Imports
from astropy.table import Table
from astropy.coordinates import SkyCoord
from astropy import units as u
```

```
# Open table files
file1 = '../data/HCG7_SDSS_photo.dat'
file2 = '../data/HCG7_2MASS.tbl'
sdss = Table.read(file1, format='ascii')
twomass = Table.read(file2, format='ascii')

# Match between catalogs
coo_sdss = SkyCoord(sdss['ra']*u.deg, sdss['dec']*u.deg)
coo_twomass = SkyCoord(twomass['ra'], twomass['dec'])
idx_sdss, d2d_sdss, d3d_sdss = coo_twomass.match_to_catalog_sky(coo_sdss)

# Print matches
print("Matched values by index: \n")
print(idx_sdss)
```

```
Matched values by index:

[368 370   6 116 255 454 501  41 174 505  13 515 624 523 338 297 389 294
 573 539 500 140 622]
```

2.12.18 tmerge

Please review the *Notes* section above before running any examples in this notebook

Tmerge is used to combine columns or rows of multiple tables. There are two Astropy Table tasks for this, `vstack` and `hstack`. We take these examples from the Astropy table docs.

```
# Astronomy Specific Imports
from astropy.table import Table, vstack, hstack
```

```
# Setup tables
obs1 = Table.read("""name    obs_date    mag_b  logLx
                     M31     2012-01-02  17.0   42.5
                     M82     2012-10-29  16.2   43.5
                     M101    2012-10-31  15.1   44.5""", format='ascii')

obs2 = Table.read("""name    obs_date    logLx
                     NGC3516 2011-11-11  42.1
                     M31     1999-01-05  43.1
                     M82     2012-10-30  45.0""", format='ascii')

# Vertical stack
print(vstack([obs1, obs2]))
```

```
  name   obs_date  mag_b logLx
------- ---------- ----- -----
    M31 2012-01-02  17.0  42.5
    M82 2012-10-29  16.2  43.5
   M101 2012-10-31  15.1  44.5
NGC3516 2011-11-11    --  42.1
    M31 1999-01-05    --  43.1
    M82 2012-10-30    --  45.0
```

```
# Setup tables
t1 = Table.read("""a   b    c
                   1   foo  1.4
                   2   bar  2.1
                   3   baz  2.8""", format='ascii')
t2 = Table.read("""d     e
                   ham   eggs
                   spam  toast""", format='ascii')

# Horizontal stack
print(hstack([t1, t2]))
```

```
 a   b   c   d     e
--- --- --- ---- -----
  1 foo 1.4  ham  eggs
  2 bar 2.1 spam toast
  3 baz 2.8   --    --
```

2.12.19 tselect-tproject-tquery

Please review the *Notes* section above before running any examples in this notebook

Tselect is used to create a new table from selected rows, tproject from selected columns, and tquery from a combination of selected rows and columns. We show two examples of how to generate a new table from selected columns and selected rows. You can combine these two pieces of code in either order to get a tquery like result. For row filtering we combine boolean masks using the Python bitwise operators. There is an alternate way to do selections if you have already organized your table into groups by using the filter method, but the user will still need to write a custom filtering function to provide to `filter`.

```
# Astronomy Specific Imports
from astropy.table import Table
```

```
# For selecting rows can use bitwise operators to generate a boolean mask
table1 = Table(dtype=ex_table.dtype)
boolean_mask = (ex_table['sname'] == 'star4') | (ex_table['radius'] == 20)

#
subset = ex_table[boolean_mask]
subset.pprint()
```

```
sname radius fwhm
----- ------ ----
star4      1 0.75
star5     20 13.0
```

```
# For selecting columns we can pull the required columns
# out of the original table with the column names
table2 = ex_table['sname','fwhm']
table2.pprint()
```

```
sname fwhm
----- ----
star1  6.5
star2  5.1
star3  0.5
star4 0.75
star5 13.0
```

2.12.20 tsort

Please review the *Notes* section above before running any examples in this notebook

Tsort, as you would guess, sorts a table. `Astropy Table` objects have a built in sort method. You can even sort by more then one column. Sorting is preformed inplace so in this example we make a copy of the table first.

```
# Standard Imports
import numpy as np

# Astronomy Specific imports
from astropy.table import Table
```

```
# Sorting
sorted_table = ex_table.copy()
sorted_table.sort('radius')
sorted_table.pprint()

print('\n')

# Reverse the sort
sorted_table.reverse()
sorted_table.pprint()

print('\n')

# Sort by more then one column
sorted_table.sort(['radius','fwhm'])
sorted_table.pprint()
```

```
sname radius fwhm
----- ------ ----
star4      1 0.75
star3      2  0.5
star2      7  5.1
star1     10  6.5
star5     20  4.5


sname radius fwhm
----- ------ ----
star5     20  4.5
star1     10  6.5
star2      7  5.1
star3      2  0.5
star4      1 0.75


sname radius fwhm
----- ------ ----
star4      1 0.75
star3      2  0.5
star2      7  5.1
star1     10  6.5
star5     20  4.5
```

2.12.21 tstat

Please review the *Notes* section above before running any examples in this notebook

Tstat gives you the mean, standard deviation, minimum and maximum of a column. This can be done by using the `Table` info function, with the 'stats' argument.

```
# Astronomy Specific Imports
from astropy.table import Table
```

```
# All column stats
ex_table.info('stats')
```

(continues on next page)

(continued from previous page)

```
print("\n")

# Specific column stats
ex_table['radius'].info('stats')
```

```
<Table length=5>
 name  mean       std       min max
------ ---- ------------- --- ---
 sname   --            --  --  --
radius  8.0 6.84105255059   1  20
  fwhm 3.47 2.41321362502 0.5 6.5


name = radius
mean = 8.0
std = 6.84105255059
min = 1
max = 20
n_bad = 0
length = 5
```

2.12.22 Not Replacing

- gtedit - Graphically edit a table. Deprecated.
- gtpar - Pset to specify graph parameters for gtedit task. Deprecated.
- keytab - Copy n image or table header keyword to a table element. See Astropy Tables documentation.
- keypar - Copy an image or table header keyword to an IRAF parameter. See Astropy FITS documentation.
- keyselect - Copy selected image header keywords to sdas table. See **images.imutil**
- parkey - Put an IRAF parameter into an image or table header keyword. See Astropy FITS documentation.
- tabkey - Copy a table element to an image or table header keyword. See the above notebook and Astropy FITS documentation.
- tcheck - Check STSDAS table element values. See Astropy Tables documentation.
- tchsize - Change allocated sizes of various sections of a table. Deprecated.
- tcreate - Create a FITS table from an ASCII descriptor table. see *tcopy-tdump* and Unified I/O documentation.
- tdelete - Delete tables. Deprecated.
- tedit - Edit a table. See Astropy Tables documentation or *partab*.
- thedit - Edit or print table header keywords. See **images.imutil.hedit**
- thselect - Print table keyword values. See **images.imutil.hselect**
- tproduct - Form the Cartesian product of two tables. See *tjoin*
- Trebin - Allows the user to rebin columns in a table using linear or spline interpolation. See the Astropy binning doc section for a subset of this functionality.
- tread - Browse through a table. See Astropy Tables documentation.
- tscopy - Copy row/column subsets of tables using selectors. See *tselect-tproject-tquery*.

- ttranspose - Transpose or flip a table. Deprecated.
- tupar - Edit table header keywords. Interactive GUI. Deprecated
- tupar - Edit table header keywords. Interactive GUI. See *tchcol*

Cosmic Ray Rejections:

2.13 noao.imred.crutil

The noao.imred.crutil package contains various algorithms for finding and replacing cosmic rays in single images or image sets.

2.13.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Contents: * *crgrow* * *crmedian*

2.13.2 crgrow

Please review the *Notes* **section above before running any examples in this notebook**

The crgrow replacement uses the `skimage.morphology` package to grow the values in any numpy array. The dilation task is a wrapper around `scipy.ndimage.grey_dilation`. You can insert any kernal type where `disk` is called in this example. See the skimage.morphology.dilation for more information. More kernel shapes are also listed on this page.

```
# Standard Imports
from skimage.morphology import disk,dilation

# Astronomy Specific Imports
from astropy.io import fits
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615006'
Observations.download_products(obsid,productFilename="iczgs3ygq_flt.fits")
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Change this value to your desired data file
test_data = './mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits'
out_file = 'crgrow.fits'

# Read in your fits file, when using some fits file, the byteswap call is required to
# make sure your array data type is correct when the dilation function is used. This
# may be due to a bug in the dilation funciton.
```

(continues on next page)

(continued from previous page)

```
# For this example we will work with the 3rd extensions, the DQ array
hdu = fits.open(test_data,mode='update')
hdu.info()
dq1 = hdu[3].data.byteswap().newbyteorder('=')

# Dilation used to grow the CR flags, here we use the disk radius 2 shape kernel
grownDQ = dilation(dq1, disk(2))

# Re-assign the changed array to our original fits file and close save the updated␣
↪FITS to a new file.
hdu[3].data = grownDQ
hdu.writeto(out_file, overwrite=True)
```

```
Filename: ./mastDownload/HST/ICZGS3YGQ/iczgs3ygq_flt.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     266   ()
  1  SCI           1 ImageHDU       140   (1014, 1014)   float32
  2  ERR           1 ImageHDU        51   (1014, 1014)   float32
  3  DQ            1 ImageHDU        43   (1014, 1014)   int16
  4  SAMP          1 ImageHDU        37   (1014, 1014)   int16
  5  TIME          1 ImageHDU        37   (1014, 1014)   float32
  6  WCSCORR       1 BinTableHDU     59   7R x 24C   [40A, I, A, 24A, 24A, 24A, 24A,␣
↪D, D, D, D, D, D, D, D, 24A, 24A, D, D, D, D, J, 40A, 128A]
```

2.13.3 crmedian

Please review the *Notes* section above before running any examples in this notebook

The crmedian task is a way to identify and replace cosmic rays in a single image by detecting pixels that deviate a statistically significant amount from the median by comparing to a median filtered version of the image. The identified cosmic rays can then be replaced by the median filtered value. A similar algorithm has been used in ccdproc.cosmicray_median. In `ccdproc.cosmicray_median` you also have the option of using an error array. If none is provided the standard deviation of the data is used. Ccdproc is an evolving package, please see their documentation for more information on usage.

```
# Astronomy Specific Imports
from astropy.io import fits
from astropy import units
from ccdproc import cosmicray_median, fits_ccddata_reader
from astroquery.mast import Observations
```

```
# Download test file using astroquery, this only needs to be run once
# and can be skipped if using your own data.
# Astroquery will only download file if not already present.
obsid = '2004615003'
Observations.download_products(obsid,productFilename="iczgs3y5q_flt.fits")
```

```
INFO: Found cached file ./mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits with expected␣
↪size 16534080. [astroquery.query]
```

```
# Change these values to your desired data files
test_data = './mastDownload/HST/ICZGS3Y5Q/iczgs3y5q_flt.fits'

```

(continues on next page)

(continued from previous page)

```
# First we need to pull out the science and error(uncertainty) array to
# create CCDData objects. Our acutal unit is electrons/sec, this is not
# accepted by the current set of units
image_data = fits_ccddata_reader(test_data, hdu=1, unit=units.electron/units.s, hdu_
↪uncertainty=2)
error_data = image_data.uncertainty.array

# Now we run cosmicray_median, since we input a CCDData type, a CCDData type is␣
↪returned
# If a numpy.ndarray if the input data type, it will return a numpy.ndarray
newdata = cosmicray_median(image_data, error_image=error_data, thresh=5, mbox=11,␣
↪rbox=11, gbox=3)
```

```
INFO: using the unit electron / s passed to the FITS reader instead of the unit␣
↪ELECTRONS/S in the FITS file. [astropy.nddata.ccddata]
```

```
/Users/ogaz/miniconda3/envs/irafdev/lib/python3.5/site-packages/ccdproc/core.py:1565:␣
↪RuntimeWarning: divide by zero encountered in true_divide
  rarr = (data - marr) / error_image
```

2.13.4 Not Replacing

- cosmicrays - Remove cosmic rays using flux ratio algorithm.
- craverage - Detect CRs against average and avoid objects.
- crcombine - Combine multiple exposures to eliminate cosmic rays.
- credit - Interactively edit cosmic rays using an image display.
- crfix - Fix cosmic rays in images using cosmic ray masks.
- crnebula - Detect and replace cosmic rays in nebular data.

Other / General Tools:

2.14 lists

List processing package.

2.14.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

The simple tasks in this package are already covered by `Astropy` or other Python built-ins

Contents:

- *lintran*
- *raverage*
- *table-unique*

2.14.2 lintran

Please review the *Notes* section above before running any examples in this notebook

The lintran task will linear transform a list of coordinates. There are various tasks in the numpy linear algebra package that can be used to achieve this in Python. Below is a simple example of a 90 degree rotation.

```
# Standard Imports
import numpy as np
```

```
# 90 degree rotation counterclockwise
x_points = np.array([1,4,5,7,3])
y_points = np.array([4,5,2,2,4])
points = np.vstack([x_points,y_points])
transform_matrix = a = np.array([[0, -1],
                                 [1, 0]])

new_points = np.linalg.inv(a).dot(points)

print("new x values: {}".format(new_points[0]))
print("new y values: {}".format(new_points[1]))
```

```
new x values: [ 4.  5.  2.  2.  4.]
new y values: [-1. -4. -5. -7. -3.]
```

2.14.3 raverage

Please review the *Notes* section above before running any examples in this notebook

raverage computes the running average and standard deviation for a 1-dimensional array. We can efficiently do this in Python using numpy's convolve function. For a more complex but more efficient implementatin of a rolling average that will also give you an output standard deviation you can use strides in `numpy` as seen in this example here.

```
# Standard Imports
import numpy as np
```

```
# setup test data
data = np.arange(20)
# kernel size
N = 4

out_data = np.convolve(data, np.ones((N,))/N, mode='valid')

print("original array: {}".format(data))
print("out_data: {}".format(out_data))
```

```
original array: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
out_data: [  1.5   2.5   3.5   4.5   5.5   6.5   7.5   8.5   9.5  10.5  11.5  12.5
  13.5  14.5  15.5  16.5  17.5]
```

2.14.4 table-unique

Please review the *Notes* section above before running any examples in this notebook

The table task a list of text and transfers it to a table. We will show an example of this using Astropy Tables and a text file with return seperated values. There are various parameters for reading in ascii files, documentation found here. We will continue this example to show how to extract that list and then apply a unique requirement, and save the list back out to a file, as in the IRAF task unique. More information on going to and from `Astropy Tables` see:

```
# Astronomy Specific Imports
from astropy.table import Table, unique
```

```
# Read text file into table
text_file = '../data/table.txt'
tab = Table.read(text_file, format='ascii.no_header')
tab.pprint()
```

```
 col1
-----
star1
star2
star3
star3
star4
star5
star5
star6
```

```
# Run unique and print table
out_table = unique(tab)
out_table.pprint()

# Save results to new text file
out_table.write("out_table.txt",format='ascii')
```

```
 col1
-----
star1
star2
star3
star4
star5
star6
```

2.14.5 Not Replacing

- average - see **images.imutil.imstatistics** or numpy tools
- columns - used to convert multicolumn data into CL lists, deprecated
- rgcursor - Read the graphics cursor, deprecated
- rimcursor - Read the image display cursor, see **images.tv** or Python imexam
- tokens - deprecated
- words - deprecated, see Python's built in file reader

2.15 noao

Top level of the noao package, only contains the task observatory.

2.15.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Observatory is the only task on the top noao module level. See the Python example below.

2.15.2 observatory

The observatory task will return various information about an observatory location. This task is included in `Astropy` as the `astropy.coordinates.EarthLocation` class. See the doc pages here and here for more information.

```
# Astronomy Specific Imports
from astropy.coordinates import EarthLocation
```

```
EarthLocation.of_site('Apache Point Observatory')
```

$$(-1463969.3, -5166673.3, 3434985.7)\ \mathrm{m}$$

2.16 stsdas.analysis.nebular

Tasks for analyzing nebular emission lines.

2.16.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

The nebular package has been replaced in Python by the PyNeb package, documentation can be found here, with a homepage here. In this notebook we will simply list the equivalent Python task for the original IRAF task where possible.

Contents:

- *abund*
- *ionic*
- *ntcontour-ntplot*
- *redcorr*
- *temden*
- *zones*

2.16.2 abund

Please review the *Notes* section above before running any examples in this notebook

abund - Derive ionic abundances relative to H+ in 3-zone nebula.

- See the getIonAbundance function and section 1.9 of the PyNeb handbook.

2.16.3 ionic

Please review the *Notes* section above before running any examples in this notebook

ionic - Determine ionic abundance relative to H+.

- see the Atom class in PyNeb

2.16.4 ntcontour-ntplot

Please review the *Notes* section above before running any examples in this notebook

ntcontour - Plot contours of N_e- or T_e-sensitive line ratios.

ntplot - Construct N_e vs. T_e plot for observed diagnostic ratios.

- See section 2.4 of the PyNeb handbook.

2.16.5 redcorr

Please review the *Notes* section above before running any examples in this notebook

redcorr - Correct line flux for interstellar reddening.

- See the RedCorr class in PyNeb.

2.16.6 temden

Please review the *Notes* section above before running any examples in this notebook

temden - Determine electron temperature or density from diagnostic ratio.

- See section 1.11 of the PyNeb handbook.

2.16.7 zones

Please review the *Notes* section above before running any examples in this notebook

zones - Determine electron temps & densities in 3-zone nebula.

- See the Atom object in PyNeb.

2.16.8 Not Replacing

- at_data - Documentation on the atomic reference data. Derecated.
- diagcols - Pset for zone temperature & density column names. Deprecated.
- fluxcols - Pset for line flux column names. Deprecated.
- nlevel - Documentation on the N-level atom approximation. Deprecated.

2.17 stsdas.analysis.statistics

The statistics package contains statistical analysis tasks.

2.17.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Many of the tasks below are documented in their respective library documentation. Please see the links provided for example usage.

Contents:

2.17.2 bhkmethod

Please review the *Notes* **section above before running any examples in this notebook**

The bhkmethod task is used to compute the generalized Kendall's tau correlation coefficient. We show a short example here taken from the scipy.stats.kendalltau documentation.

```
# Standard Imports
from scipy import stats
```

```
x1 = [12, 2, 1, 12, 2]
x2 = [1, 4, 7, 1, 0]
tau, p_value = stats.kendalltau(x1, x2)
print("tau: {}".format(tau))
print("p_value: {}".format(p_value))
```

```
tau: -0.4714045207910316
p_value: 0.2827454599327748
```

2.17.3 buckleyjames-kmestimate

Please review the *Notes* section above before running any examples in this notebook

The buckleyjames and kestimate tasks compute linear regression coefficients and esitmators with the Kaplan-Meier estimator. There is currently a Python package called `lifelines` that have this fitter.

2.17.4 coxhazard

Please review the *Notes* section above before running any examples in this notebook

The coxhazard task is used to compute the correlation probability by Cox's proportional hazard model. See an example of this fitter in the lifelines package.

2.17.5 kolmov

Please review the *Notes* section above before running any examples in this notebook

The kolmov task uses the Kolmogorov-Smirnov test for goodness of fit. You can find both the one-sided and two-sided test in `scipy`:

- one-sided
- two-sided

2.17.6 spearman

Please review the *Notes* section above before running any examples in this notebook

The spearman task is used to compute regression coefficients by Scmitt's method. `Scipy` contains a version of this task, see documentation here.

```
# Standard Imports
from scipy import stats
```

```
rho, pvalue = stats.spearmanr([1,2,3,4,5],[5,6,7,8,7])
print("rho: {}".format(rho))
print("p-value: {}".format(pvalue))
```

```
rho: 0.8207826816681233
p-value: 0.08858700531354381
```

2.17.7 twosampt

Please review the *Notes* section above before running any examples in this notebook

The twosampt task is used to determine if two sets of data are from the same population. It provided the following types of two sample test: geham-permute, gehan-hyper, logrank, peto-peto, and peto-prentice. These tests do not currently have an equivalent in Scipy, but the following two sample tests are availalbe:

- Ranksums
- Wilcoxon
- Man-Whitney

2.17.8 Not Replacing

- censor - Information about the censoring indicator in survival analysis. Deprecated.
- emmethod - Compute linear regression for censored data by EM method. Deprecated.
- schmittbin - Compute regression coefficients by Schmitt's method. Deprecated.
- survival - Provide background & overview of survival analysis. Deprecated.

2.18 stsdas.toolbox.tools

General utilities

2.18.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

Contents:

2.18.2 base2dec-dec2base

Please review the *Notes* **section above before running any examples in this notebook**

The base2dec and dec2base tasks transform strings to decimal integers, and decimal integers to other base strings. Python has various built ins for these conversion. The `int()` function can transform any base to integer, which can then be printed in decimal. For the reverse direction Python contains built in functionality to transform integers to octal, hexidecimal, and binary.

- oct function
- hex function

```
# base 16 to integer
a = int("b1", base=16)
print(a)

# integer to base 16
b = hex(177)
print(b)

# integer to binary
c = "{0:b}".format(177)
print(c)
```

```
177
0xb1
10110001
```

2.18.3 ddiff

Please review the *Notes* section above before running any examples in this notebook

Ddiff is used to print differences between two directory trees. This can be replicated using os.walk and a little bit of set maniputaion.

```
# Standard Imports
import os
```

```
full_filepaths1 = []
full_filepaths2 = []

# loop through walk iterator, using current directory
# for this example with the sting "."
for root, dirs, files in os.walk("."):
    for filestring in files:
        full_filepaths1.append(os.path.join(root,filestring))

# We will use the same directory for this example, so the set difference should be␣
↪empty
for root, dirs, files in os.walk("."):
    for filestring in files:
        full_filepaths2.append(os.path.join(root,filestring))

# Now we turn both filepath lists into sets, and take the difference
set1 = set(full_filepaths1)
set2 = set(full_filepaths2)
print(set1 - set2)
```

```
set()
```

2.18.4 epoch-tepoch

Please review the *Notes* section above before running any examples in this notebook

Epoch and tepoch are used to convert time formats. This functionality is heavily covered by the Astropy Time module and the documentation is well developed. Alternatively, there is a native Python datetime module. Please see the linked documentation for more thorough details.

Below we will show an example of how to combine the Astropy `Time` module with the `Table` module. This example uses `Table` mixin columns. Before expanding on this example for your own use, please read over the mixin column documentation.

```
# Astronomy Specific Imports
from astropy.time import Time
from astropy.table import Table
```

```
# Here we setup a simple Astropy Table, and attach some dates
# The Time wrapper around the epoch variable is for the Astropy
# Time object.
objname = ['obj1', 'obj2']
epoch = Time(['2010-1-2', '2010-1-3'])
tab = Table([objname, epoch], names=['name', 'epoch'])

# And here in a single line we can display this object in mjd
mjd = tab['epoch'].mjd
print("mjd: {}\n".format(mjd))

# To make this change permanent we can re-assign the whole column
tab['epoch'] = mjd

# Print updated Table column
print(tab['epoch'])
```

```
mjd: [ 55198.  55199.]

 epoch
-------
55198.0
55199.0
```

2.18.5 fparse

Please review the *Notes* section above before running any examples in this notebook

Fparse is used to parse file specifications and leave results in parameters. This can be done using the `os` path.split function and the built in String split method.

```
# Standard Imports
import os
```

```
# code goes here
my_filepath = "/home/user/snowball/stars.txt"
directory, filename = os.path.split(my_filepath)
print(directory)
print(filename)
print(filename.split("."))
```

```
/home/user/snowball
stars.txt
['stars', 'txt']
```

2.18.6 tprecess

Please review the *Notes* section above before running any examples in this notebook

Tprecess is used to precess images, tables, or lists of coordinates. This capability is part of the Astropy coordinates package. Please explore the doumentation for more instruction. In particular, see the second example in this section for a transformation example. For a larger overview of how coordinates are handeled in Astropy please start at the top of this documentation page.

2.18.7 Not Replacing

- mkapropos - Make the apropos database. Deprecated.
- newredshift - Change the redshift of spectra.
- uniqfile - Give a file a unique name prior to archiving. Deprecated.
- uniqid - Create a unique character string identifier. Deprecated.
- uniqname - Create a unique file name for archiving. Deprecated.
- uniqtab - Give all the files in an STSDAS table unique names. Deprecated.

2.19 utilities

A miscellaneous utilities package.

2.19.1 Notes

For questions or comments please see our github page. **We encourage and appreciate user feedback.**

Most of these notebooks rely on basic knowledge of the Astropy FITS I/O module. If you are unfamiliar with this module please see the Astropy FITS I/O user documentation **before using this documentation**.

The utilities package can be easily replicated with Python functionality.

Contents:

- *detab-entab*
- *lcase-ucase*
- *urand*

2.19.2 detab-entab-translit

Please review the *Notes* section above before running any examples in this notebook

Detab and entab are used to replace tabs with blanks or vice versa. Python contains a built-in replace method which can be used for this purpose.

2.19.3 lcase-ucase

Please review the *Notes* section above before running any examples in this notebook

lcase and ucase are used to convert a file to lower case or uppercase. Python contains built in functionality for string replacement, including `lower` and `upper` methods.

2.19.4 urand

Please review the *Notes* section above before running any examples in this notebook

Urand provides a uniform random number generator. This utility is contained in `Numpy` with the numpy.randome.uniform task. `Numpy` also contains other types of random generation.

2.19.5 Not Replacing

- curfit - Fit data with Chebyshev, Legendre or spline curve. See **images.imfit.it1d-lineclean**
- polyfit - Fit polynomial to list of X,Y data. See **images.imfit.fit1d-lineclean**
- surfit - Fit a surface, z=f(x,y), to a set of x, y, z points. See **images.imfit.imsurfit**
- split - Split a large file into smaller segments. Deprecated.

2.20 Index

To search IRAF tasks by task name, see the index linked below.

2.20.1 Jupyter Notebook Index

Index