nickel-calib

October 21, 2023

1 Nickel Direct Imaging Data Reduction

Basis of this jupyter notebook is from Keerthi Vasan Gopala Chandrasekaran (UC-Davis), who created it from Elinor Gates' (UCO/Lick) 2018 Observational Astronomy Workshop python data reduction activity. Additional code contributions and conversion so it would work under Python 3 were from Azalee Bostroem (UC-Davis). Elinor Gates subsequently added commentary, expanded the code to make sure everything is done inside python, and added the cosmic ray rejection section. Jon Rees modified things further and added the photometry section. This is designed to work with Python 3.

If the data are properly acquired and FITS headers are accurate, this should work as a basic data reduction pipeline. However, proceeding slowly, one step at a time, examining calibration and image frames at each step is encouraged so that understanding of each step and its importance to the general data reduction is understood, as well as catching errors and implementing fixes as soon as possible in the procedures.

1.1 Import the Necessary Python Packages

[ ]:
```
from astropy.io import fits,ascii
import numpy as np
import sys, getopt,os
from glob import glob
import math
# shutil is used for the file copying
import shutil
# tqdm gives us a handy progress bar for some of the more time consuming steps
from tqdm.notebook import tqdm
```

1.2 Deal with Astroscrappy

Later in the notebook we'll use astroscrappy to deal with cosmic ray removal. This cell will install astroscrappy if it is not already installed. If the cell runs without errors, you're (probably) good to go.

If Astroscrappy segfaults later in this notebook, try removing it before re-running the below code (pip uninstall -y astroscrappy)

If you're running this on Windows, you'll need Microsoft's Visual C++ Build Tools: https://visualstudio.microsoft.com/visual-cpp-build-tools/

If you're running on Windows Subsystem for Linux you'll need to install gcc

```
[ ]: try:
         import astroscrappy
         print("module 'astroscrappy' is installed")
     except ModuleNotFoundError:
         print("module 'astroscrappy' is not installed")
         !{sys.executable} -m pip install astroscrappy
```

1.3 Organise Data

Everyone will have their own preferred method of organising their data. As the current arbiter of this reduction activity, the notebook has been written to conform to my preferred method: Original data files should never be overwritten/altered. You should be able to go back and re-run the reduction multiple times from your original files as you learn new quirks of the data.

We will set the path to the parent directory below using the variable source_dir.

Set up initial directories

Create two directories, 'Data' and 'Reduced'. Place your data for the given night inside the 'Data' directory.

The initial directory structure should include a 'Data' directory and a 'Reduced' directory. The initial data files will be copied from 'Data' to 'Reduced', and all subsequent operations will be performed on the data in the 'Reduced' directory. If you need to re-run the reduction, you can delete the files in the 'Reduced' directory without worry.

Below we automatically set up directories for file sorting inside the Reduced directory:

Change Source Directory

You'll want to change the source directory appropriately for your data location.

```
[ ]: # The source directory. Set this to wherever your Data and Reduced directories␣
      ↪live.
     source_dir = '/home/jrees/DataReduction/20230720/'

     # Location of the folder containing all the data files (bias, domeflats,␣
      ↪twilight flats, and the data files)
     data_dir = source_dir + 'Data/'
     redu_dir = source_dir + 'Reduced/'

     # Check that the data directory actually exists
     if os.path.exists(data_dir) == False:
         raise ValueError("Data directory does not exist")
     # And make sure that it is not empty
     if len(os.listdir(data_dir)) == 0:
         raise ValueError("Data directory is empty")
```

```
# Make some directories to organise files by type (bias, flats etc.)
# The archive directory will store files that are no longer needed for the data␣
 ↪reduction,
# but still available to examine if needed if there are issues with the data␣
 ↪reduction.

biasdir = redu_dir+'Bias/'
datadir = redu_dir+'Data_files/'
domeflatdir = redu_dir+'Flat_dome/'
twiflatdir = redu_dir+'Flat_twilight/'
archivedir = redu_dir+'Archive/'


os.makedirs(biasdir,exist_ok=True)
os.makedirs(datadir,exist_ok=True)
os.makedirs(domeflatdir,exist_ok=True)
os.makedirs(twiflatdir,exist_ok=True)
os.makedirs(archivedir,exist_ok=True)
```

1.4 Copy Data

Now we copy the initial data to the Reduced directory. This way our original data remains safe, and we can always easily reproduce what we did to reduce the data.

Set any files to be removed

If you have any bad data frames, you can remove them by adding the frame numbers to delfilelist and setting delfiles = 'yes'

```
[ ]: # Copy all of the data from the Data directory to the Reduction directory
     # Our input list is just all of the FITS files in the Data directory
     ifilelist = glob(data_dir+'*.fits')
     # Our output location is the Reduced direcory so doesn't need a list

     # Copy the files using shutil
     for file in ifilelist:
         shutil.copy2(file, redu_dir)

     # And if you want to remove any known-bad files, add them to the list below and␣
      ↪set delfiles = yes
     delfiles = 'no'

     if delfiles == 'yes':
         # Set the frame numbers of the frames to delete from the Reduced directory
         delfilelist = ('d1036', 'd1037', 'd1041', 'd1045')
         for file in delfilelist:
             # Check if the file exists
             if os.path.isfile(redu_dir + file + '.fits'):
```

```
        # If it does exist, delete it
            os.remove(redu_dir + file + '.fits')
            print("Deleting FITS file " + file + '.fits' + " from Reduced␣
↪direcory.")
            print("-----------------------------------------------------")
```

1.5 Overscan Subtraction

The overscan region(s) are additional columns appended to the data that measure the overall bias values for each row at the time the data were acquired. Depending on the camera, these may be very stable over a night of observing, or vary from image to image. This bias level needs to be subtracted before further data reduction steps should be done. Overscan subtraction is done on all calibration and science files.

The original overscanLickObsP3.py code is available on-line via our optical instrument manuals will read a list of files in, determine the overscan and data regions for each file, fit the overscan, then subtract it from the data, writing out a new overscan subtracted image for each input image. This code is specific to Lick Observatory data and keywords, but could easily be altered with the appropriate keywords to work with other detectors with one or two amplifiers. The code below has been modified from the original to create input and output filelists according to the file directory denoted above. If you have a lot of data, this may take a few minutes to run.

```
[ ]: # set fit = 'yes' to do legendre fit to overscan regions, 'no' to just use the␣
     ↪median
     fit = 'yes'

     # for i in range(0,numifiles):
     #         ifile=ifilelist[i]
     #         basename=os.path.basename(ifile)
     #         print(basename)

     # Our input file list is everything in the Reduced directory
     ifilelist = glob(redu_dir+'*.fits')

     # For each file in ifilelist, we need to read in the file,
     # figure out overscan and data regions, fit the overscan with desired function␣
     ↪(if any),
     # subtract the overscan from the data, and finally write the data to an output␣
     ↪file.

     for ifile in tqdm(ifilelist):
         # The output files will have _os appended (overscan subtracted) in their␣
     ↪file names
         ofile = ifile[:-5]+ '_os.fits'
         # Read in the input FITS file using the fits module from astropy.io
         data, header = fits.getdata(ifile,header=True)
         # Change data to float
```

```
data=data.astype('float32')

# read necessary keywords from fits header

#number of pixels in image
xsize = header['NAXIS1']
ysize = header['NAXIS2']
#start column and row
xorig = header['CRVAL1U']
yorig = header['CRVAL2U']
#binning and direction of reading pixels
cdelt1 = header['CDELT1U']
cdelt2 = header['CDELT2U']
# number of overscan rows/columns
rover = header['ROVER']
cover = header['COVER']
#unbinned detector size
detxsize = header['DNAXIS1']
detysize = header['DNAXIS2']
#number of amplifiers
ampsx = header['AMPSCOL']
ampsy = header['AMPSROW']

# determine number and sizes of overscan and data regions
namps = ampsx*ampsy
if rover > 0:
    over=rover
    sys.exit('Program does not yet deal with row overscans. Exiting.')
else:
    over = cover
if over == 0:
    sys.exit('No overscan region specified in FITS header. Exiting.')

# single amplifier mode (assumes overscan is the righmost columns)
if namps == 1:
    biassec = data[:,xsize-cover:xsize]
    datasec = data[0:,0:xsize-cover]

    # median overscan section
    bias=np.median(biassec, axis=1)

    # legendre fit
    if fit == 'yes':
        # fit
        lfit = np.polynomial.legendre.legfit(range(0,len(bias)),bias,3)
        bias = np.polynomial.legendre.legval(range(0,len(bias)),lfit)
        # subtract overscan
        datanew = datasec
        for i in range(datasec.shape[1]):
            datanew[:,i] = datasec[:,i]-bias

    # two amplifier mode (assumes both amplifer overscans are at rightmost␣
↪columns)
    if namps == 2:
        biasseca = data[:,xsize-cover*2:xsize-cover]
        biassecb = data[:,xsize-cover:xsize]

        # median overscan sections
        biasa=np.median(biasseca,axis=1)
        biasb=np.median(biassecb,axis=1)

        # legendre fit
        if fit == 'yes':
            lfita = np.polynomial.legendre.legfit(range(0,len(biasa)),biasa,3)
            lfitb = np.polynomial.legendre.legfit(range(0,len(biasb)),biasb,3)
            biasa = np.polynomial.legendre.legval(range(0,len(biasa)),lfita)
            biasb = np.polynomial.legendre.legval(range(0,len(biasb)),lfitb)

        # Extract data regions

        # determine boundary between amplifiers
        bd=detxsize/2/abs(cdelt1)

        # calculate x origin of readout in binned units if cdelt1 negative or␣
↪positive
        if cdelt1 < 0:
            #if no binning x0=xorig-xsize-2*cover, with binning:
            x0=xorig/abs(cdelt1)- (xsize-2*cover)
        else:
            x0=xorig/cdelt1

        xtest=x0+xsize-cover*2 # need to test if all data on one or two␣
↪amplifiers

        # determine which columns are on which amplifier and subtract proper␣
↪overscan region

        if xtest < bd: # all data on left amplifier
            datanew=data[:,0:xsize-cover*2]
            m=datanew.shape[1]
            for i in range(0,m):
                datanew[:,i]=datanew[:,i]-biasa
        if x0 >= bd: # all data on right amplifier
            datanew=data[:,0:xsize-cover*2]
            m=datanew.shape[1]
            for i in range(0,m):
                datanew[:,i]=datanew[:,i]-biasb

        if xtest >= bd and x0 < bd:  #data on both amplifiers
            x1=int(bd-x0)
            dataa=data[:,0:x1]
            datab=data[:,x1:-cover*2]
            ma=dataa.shape[1]
            mb=datab.shape[1]
            for i in range(0,ma):
                dataa[:,i]=dataa[:,i]-biasa
            for i in range(0,mb):
                datab[:,i]=datab[:,i]-biasb
            # merge dataa and datab into single image
            datanew=np.hstack([dataa,datab])

    if namps > 2:
        sys.exit('Program does not yet deal with more than two overscan regions.
 ↪ Exiting.')

    # add info to header
    header['HISTORY'] = 'Overscan subtracted'

    # write new fits file
    fits.writeto(ofile,datanew,header,overwrite=True)
    # And move the input file to the archive directory
    basename=os.path.basename(ifile)
    os.rename(ifile,archivedir+basename)

# When done with the subtraction, let us know
print("Overscan subtraction completed.")
print("------------------------------------------------------")
```

2 Organise Overscan Subtracted Files

Move all the overscan subtracted (e.g. the newly created *_os.fits) bias, data, and flat field files into separate folders, based upon the OBJECT field in the FITS headers.

Check Flat Field headers

If your flat field OBJECT field does not use dome/twi for dome flats and twilight flats respectively, you will need to update the 'if' loops below.

```
[ ]: # Make a list of all the overscan subtracted files
os_files = glob(redu_dir+'*os.fits')

# Move calibration frames to appropriate directories
# In this case we are assuming that twilight flats have 'twi' in the OBJECT␣
 ↪FITS header keyword,
# dome flats have 'dome' in OBJECT, etc.  If the names are different, you'll␣
 ↪need to adjust the search strings
# for sorting the files.
for ifile in os_files:
    hdr = fits.getheader(ifile)
    basename=os.path.basename(ifile)
    if 'twi' in hdr['OBJECT'].lower():
        os.rename(ifile,twiflatdir+basename)
    elif 'dome' in hdr['OBJECT'].lower():
        os.rename(ifile,domeflatdir+basename)
    elif 'bias' in hdr['OBJECT'].lower():
        os.rename(ifile,biasdir+basename)
    else:
        os.rename(ifile,datadir+basename)
```

2.1 Create Master Bias File

The Master Bias file is the median combined bias frames. If the detector is particularly flat with no bias structure, this step may not be needed. In the case of the Nickel Direct Imaging CCD, there is significant bias structure that needs to be removed, so this step is necessary to remove that structure.

```
[ ]: # Create list of bias files
biasfiles = glob(biasdir + '*.fits')

data_stack = []
for file in biasfiles:
    data_stack.append(fits.getdata(file))



# Median combine the bias files to create the Master Bias frame
medianBias = np.median(data_stack,axis=0)

# Write out the master bias file with updated FITS header information
header = fits.getheader(biasfiles[0])
header['HISTORY'] = 'Median combined'
fits.writeto(redu_dir+'bias.fits',medianBias,header)
# For Windows machines we have to reset the data stack, otherwise it keeps the␣
 ↪bias
# files open and we can't move them
data_stack = []
```

```
# Move the no longer needed overscan subtracted frames to the archive directory
for file in biasfiles:
    basename=os.path.basename(file)
    os.rename(file,archivedir+basename)

print("Created Master Bias frame.")
print("-----------------------------------------------------")
```

2.2 Check Master Bias File

It is best to check the bias.fits file to make sure it looks OK before continuing. DS9 is a frequently used tool in astronomy for examining FITS images. DS9 is not a python tool, but freely downloadable for virtually all computer operating systems. Typical Nickel bias images look like the following.

3 Bias Subtract Flat Field and Data Frames

Because the files were sorted into subdirectories, we'll be doing essentially the same steps for the files in the Data_files, flat_dome, and flat_twilight directories.

[ ]:
```
# Bias subtracting the data files

# Make list of input files
datafilesin = glob(datadir + '*.fits')

for ifile in tqdm(datafilesin):
    # _bs stands for bias subtracted in the output file names
    ofile = ifile[:-5]+ '_bs.fits'
    data,header = fits.getdata(ifile,header=True)
    dataout = data - medianBias
    header['HISTORY'] = 'Bias subtracted'
    fits.writeto(ofile,dataout,header)
    # Again, clear the arrays so Windows doesn't complain about open files
    data = []
    header = []
    # Move the no longer needed overscan subtracted files to the archive␣
 ↪directory
    basename=os.path.basename(ifile)
    os.rename(ifile,archivedir+basename)

print("Debiased Data frames.")
print("-----------------------------------------------------")
```

[ ]:
```
# Bias subtracting the dome flat files

# Make list of input dome flat field files
datafilesin = glob(domeflatdir + '*.fits')
```

```
for ifile in tqdm(datafilesin):
    # _bs stands for bias subtracted in the output file names
    ofile = ifile[:-5]+ '_bs.fits'
    data,header = fits.getdata(ifile,header=True)
    dataout = data - medianBias
    header['HISTORY'] = 'Bias subtracted'
    fits.writeto(ofile,dataout,header)
    # Move the no longer needed overscan subtracted files to the archive␣
 ↪directory
    data = []
    header = []
    basename=os.path.basename(ifile)
    os.rename(ifile,archivedir+basename)

print("Debiased Dome Flat frames.")
print("------------------------------------------------------")
```

[ ]:
```
# Bias subtracting the twilight flat files

# Make list of input twilight flat field files
datafilesin = glob(twiflatdir + '*.fits')

for ifile in tqdm(datafilesin):
    # _bs stands for bias subtracted in the output file names
    ofile = ifile[:-5]+ '_bs.fits'
    data,header = fits.getdata(ifile,header=True)
    dataout = data - medianBias
    header['HISTORY'] = 'Bias subtracted'
    fits.writeto(ofile,dataout,header)
    # Move the no longer needed overscan subtracted files to the archive␣
 ↪directory
    data = []
    header = []
    basename=os.path.basename(ifile)
    os.rename(ifile,archivedir+basename)

print("Debiased Twilight Flat frames.")
print("------------------------------------------------------")
```

4 Create Normalised Flat Field frames

For this example we will use the twilight flat field frames, as they are generally superior to dome flats. One uses dome flats if the twilight flat field frames were unattainable due to weather or there was some other technical issues. First we will create lists of files for each filter, then combine the frames to create the final normalised flat field frame for each filter.

Choose Twilight Flats or Dome Flats and set the correct Filters

If you need to use Dome Flats instead of Twilight Flats, change flat_dir to reflect this.

By default we assume B, V, R, I filters were used. Update them below if you used a different set.

```
[ ]: # This assumes that the B, V, R, and I filters were used.  If different filters␣
     ↪were used, you'll need to change the
     # code below accordingly

     # If you are using dome flat fields, rather than twilight flats, change the␣
     ↪below to domeflatdir
     flat_dir = twiflatdir
     # If you are using different filters, or a different number of filters, set␣
     ↪them below
     filters = ['B','V','R','I']

     print("Using Filters: ")
     print (filters)
     if flat_dir == twiflatdir:
         print("Using Flat Field frames from Twilight Flat directory.")
     elif flat_dir == domeflatdir:
         print("Using Flat Field frames from Dome Flat directory.")
     else:
         print("Unrecognised Flat Field directory")
     print("-----------------------------------------------------")

     # Make list of all the flat field files in the flat field directory
     flatlist = glob(flat_dir + '*.fits')

     # Our file lists will be contained in a dictionary called flist
     flist = {}

     for filter in filters:
         # Create an empty list for the file names
         flist[filter] = []

     # Sort files into lists based on the filter used
     for ifile in flatlist:
         # Read the header for each flat file
         hdr = fits.getheader(ifile)
         # Read which filter this file was taken with
         filt = hdr['FILTNAM']
         # Loop through each of the filters in our filter set
         for filter in filters:
```

```
        # If the filter listed in the header matches the filter for this array,␣
↪add the file to the array
        if filt == filter:
            flist[filter].append(ifile)
```

[ ]:
```
# For each filter, we're going to create a Master Flat.
# We'll use another dictionary to keep track of the data stacks for each␣
↪filter,
# and one to store the normalised flat
flat_stack = {}
flat = {}

for filter in filters:
    # Initialise the stack for this filter
    flat_stack[filter] = []
    flat[filter] = []

    # Read in each file in this filter and divide by the median to normalise
    for file in flist[filter]:
        data,header = fits.getdata(file,header=True)
        data = data / np.median(data)
        # Append the data to the stacked data
        flat_stack[filter].append(data)
        # Move the now no longer needed files to archivedir
        basename=os.path.basename(file)
        os.rename(file,archivedir+basename)

    # Median combine the flat fields
    flat[filter] = np.median(flat_stack[filter],axis=0)
    # And divide by the mean to normalise
    flat[filter] = flat[filter]/np.mean(flat[filter])
    # Note in the header what we have done
    header['HISTORY'] = 'Combined and normalised flat field'
    fits.writeto(redu_dir + filter + 'flat.
↪fits',flat[filter],header,overwrite=True)
    print("Created normalised flat field in " + filter + " filter.")
    print("-----------------------------------------------------")
```

5 Check Normalised Flat Field Frames

It is wise to check the normalised flat field frames using DS9 or similar tool. Most pixel values should be very close to 1.0. A typical B-band normalised flat field is shown as an example.

6 Flat Field Data Frames

Flat fielding data is an essential step in the data reduction to calibrate the relative sensitivities of each pixel. First the data files will be sorted based on their filters, then each frame divided by the normalised flat field file in the appropriate filter.

```
[ ]: # Our file lists will again be contained in a dictionary called flist
     flist = {}

     for filter in filters:
         # Create an empty list for the file names
         flist[filter] = []

     # Make list of all bias subracted data files
     datalist = glob(datadir + '*.fits')

     # Sort files into lists based on the filter used
     for ifile in datalist:
         # Read the header for each file
         hdr = fits.getheader(ifile)
         # Read which filter this file was taken with
         filt = hdr['FILTNAM']
         # Loop through each of the filters in our filter set
         for filter in filters:
             # If the filter listed in the header matches the filter for this array,␣
      ↪add the file to the array
             if filt == filter:
                 flist[filter].append(ifile)
```

```
[ ]: # For each filter, we're going to divide the Data by the Master Flat.
     # We'll use another dictionary to keep track of the data stacks for each filter
     data_stack = {}

     for filter in filters:
         # Initialise the stack for this filter
         data_stack[filter] = []

         # Read in each file in this filter and divide by the relevant flat
         for file in tqdm(flist[filter]):
             data,header = fits.getdata(file,header=True)
             dataout = data / flat[filter]
             # Note in the header what we have done
             header['HISTORY'] = 'Flat Fielded'
             # Create the output filename
             ofile = file[:-5]+ '_ff.fits'
             # And write out the file
             fits.writeto(ofile,dataout,header)
             # Move bias subtracted images to archive
```

```
        data = []
        header = []
        basename = os.path.basename(file)
        os.rename(file,archivedir+basename)

    print("Flatfielded data frames in " + filter + " filter.")
    print("-----------------------------------------------------")
```

7 Examine Flat Fielded Images

It is highly recommended to examine all the images after flat fielding to be sure that the flat field correction has been done propertly. The image below shows a properly flat fielded image.

8 Fix Known Bad Columns in Nickel CCD2 Images

The Nickel CCD2 detector has a number of known bad columns (easily seen in the flat fielded image above). These columns can be "fixed" by replacing them with the mean values of neighboring columns. First a bad pixel pixel mask is made highlighting the known bad columns. Then for each bad pixel, the mean of the surrounding good pixels is calculated and replaces the bad pixel. This procedure is somewhat time consuming, so be patient while it runs. Do not be alarmed if it gives a warning about converting mask elements to nan, as it still works correctly.

[ ]:
```
# Procedure to fix known bad columns in CCD2 images.  2016 Oct 2 E. Gates

# Create list of flat fielded data
datalist = glob(datadir + '*.fits')

# _bp in output file name stands for bad pixel corrected
#dataout = [i[:-5]+ '_bp.fits' for i in datain]

#n=len(datain)
# size of box for area around bad pixel to be averaged
s=2

# read in one image to get image size for bad pixel mask
data,header=fits.getdata(datalist[0],header=True)

# make bad pixel mask
mask=np.ma.make_mask(data,copy=True,shrink=True,dtype=bool)
mask[:,:]=False
mask[:,255:257]=True
mask[:,783:785]=True
mask[:,1001:1003]=True

# loop for all the data bad pixel correction
# Progress bar because this can take a looong time
```

```
for file in tqdm(datalist):
    data,header=fits.getdata(file,header=True)
    mdata=np.ma.masked_array(data,mask=mask,fill_value=np.nan)
    dataFixed=data.copy()
    for i in range(0,mdata.shape[0]):
        for j in range(0,mdata.shape[1]):
            if math.isnan(mdata[i,j]):
                x1=i-s
                x2=i+s+1
                y1=j-s
                y2=j+s+1
                if x1<0:
                    x1=0
                if x2>mdata.shape[0]:
                    x2=mdata.shape[0]
                if y1<0:
                    y1=0
                if y2>mdata.shape[1]:
                    y2=mdata.shape[1]
                dataFixed[i,j]=np.mean(mdata[x1:x2,y1:y2])
    header['HISTORY']='Bad columns replaced'
    ofile = file[:-5]+ '_bp.fits'
    fits.writeto(ofile,dataFixed,header)
    # Move the now no longer needed files to archivedir
    data = []
    header = []
    mdata = []
    basename=os.path.basename(file)
    os.rename(file,archivedir+basename)

print("Fixed bad columns in Data frames.")
print("------------------------------------------------------")
```

9 Examine Bad Pixel Corrected Images

As always, it is good to check the pixel corrected images using DS9 or other image display tool. You can see in the image below that the bad columns were fixed reasonably well.

10 Cosmic Ray Removal

While the data probably look very good at this point, there are likely many cosmic rays contaminating the data. Removing all cosmic rays with software is difficult, but there are scripts that do a pretty good job. In this case we'll use the python module astroscrappy to do cosmic ray rejection. If you don't have astroscrappy installed, you'll want to install it using pip:

pip install astroscrappy

Note, it is not unusual to have to hand remove cosmic rays that are contaminating key pixels for data analysis, but that won't be covered in this jupyter notebook.

[ ]:
```
import astroscrappy
# Make a list of all the reduced data files
datalist = glob(datadir + '*.fits')
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

for file in tqdm(datalist):
    data,header=fits.getdata(file,header=True)
    data_fixed = data.copy()
    mask = np.ma.make_mask(data,copy=True,shrink=True, dtype=np.bool_)
    mask[:,:] = False
    crmask,dataCR = astroscrappy.
 ↪detect_cosmics(data_fixed,inmask=mask,cleantype='medmask')
    header['HISTORY'] = 'CR and bad pixels fixed with astroscrappy'
    ofile = file[:-5]+ '_crj.fits'
    fits.writeto(ofile,dataCR,header)
    # Move the now no longer needed files to archivedir
    data = []
    header = []
    basename=os.path.basename(file)
    os.rename(file,archivedir+basename)



print("Completed Cosmic Ray Removal using astroscrappy.")
print("------------------------------------------------------")
```

11 Inspect Final Images

We have reached the end of the basic data reduction procedure where we have performed overscan subtraction, bias subtraction, flat field correction, removed the bad pixels in the CCD, and replaced cosmic ray hits. The saved final images can now be analyzed for whatever science goal is desired, e.g. astrometry or photometry.

12 Additional Python Resources and Tutorials

Python4Astronomers http://python4astronomers.github.io/intro/intro.html

AstroPython Tutorials http://www.astropython.org/tutorials/

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

Python for Astronomers http://www.iac.es/sieinvens/siepedia/pmwiki.php?n=HOWTOs.EmpezandoPython

13 Making Three Color Images with DS9

Now that you have the data reduced, you can make a pretty three color image. Basic usage of DS9 RGB frames is described in the following video.

https://www.youtube.com/watch?v=G77RcsAfMGM

14 Making Three Color Images with GIMP

Basic tutorial to make a three color images with GIMP.

https://www.youtube.com/watch?v=56-ZaZbA3S0

15 Analyzing Data

Imexam is a convient tool based on IRAF IMEXAMINE. One can do aperture photometry, radial profile plots, FWHM measurements, etc. with this tool. Instructions for installation and use are available on-line at https://imexam.readthedocs.io/en/0.9.1/

Other photometry tools are part of the photutils python package (in fact some of the imexam procedures require photoutils). https://photutils.readthedocs.io/en/stable/