Advanced Astronomical Observations 2021
Session 14: Basic CCD Data Reduction 3

Kinoshita Daisuke

07 May 2021
publicly accessible version

About this file...

- Important information about this file
  - The author of this file is Kinoshita Daisuke.
  - The original version of this file was used for the course "Advanced Astronomical Observations" (course ID: AS6005) offered at Institute of Astronomy, National Central University from February 2021 to June 2021.
  - The file is provided in the hope that it will be useful, but there is no guarantee for the correctness. Use this file at your own risk.
  - If you are willing to use this file for your study, please feel free to use. I'll be very happy to receive feedback from you.
  - If you are willing to use this file for your teaching, please contact to Kinoshita Daisuke. When you use this file partly or entirely, please mention clearly that the author of the original version is Kinoshita Daisuke. Please help me to improve the contents of this file by sending your feedback.
  - Contact address: `https://www.instagram.com/daisuke23888/`

We do CCD data reduction for upcoming sessions.

1 Downloading data

Download a set of data.

```
% curl -k -o data_ao2021_s13.tar.xz \
? https://s3b.astro.ncu.edu.tw/advobs_202102/data/data_ao2021_s13.tar.xz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1645M  100 1645M    0     0  3254k      0  0:08:37  0:08:37 --:--:-- 4226k
% ls -l data_ao2021_s13.tar.xz
-rw-r--r--  1 daisuke  taiwan  1725532936 May  5 00:53 data_ao2021_s13.tar.xz
```

2 Extracting data

Extract the data. 534 FITS files are extracted from the tar archive file.

```
% tar xJvf data_ao2021_s13.tar.xz
x data_ao2021_s13/
x data_ao2021_s13/lot_20210214_0085.fits
x data_ao2021_s13/lot_20210214_0086.fits
x data_ao2021_s13/lot_20210214_0087.fits
x data_ao2021_s13/lot_20210214_0088.fits
```

```
x data_ao2021_s13/lot_20210214_0089.fits

.....

x data_ao2021_s13/lot_20210214_0967.fits
x data_ao2021_s13/lot_20210214_0968.fits
x data_ao2021_s13/lot_20210214_0969.fits
x data_ao2021_s13/lot_20210214_0970.fits
x data_ao2021_s13/lot_20210214_0971.fits
% ls -l data_ao2021_s13 | head
total 4306
-rw-r--r--  1 daisuke  taiwan  8398080 Feb 15 00:24 lot_20210214_0085.fits
-rw-r--r--  1 daisuke  taiwan  8398080 Feb 15 00:25 lot_20210214_0086.fits
-rw-r--r--  1 daisuke  taiwan  8398080 Feb 15 00:26 lot_20210214_0087.fits
-rw-r--r--  1 daisuke  taiwan  8398080 Feb 15 00:28 lot_20210214_0088.fits
-rw-r--r--  1 daisuke  taiwan  8398080 Feb 15 00:29 lot_20210214_0089.fits
-rw-r--r--  1 daisuke  taiwan  8400960 Feb 15 00:36 lot_20210214_0093.fits
-rw-r--r--  1 daisuke  taiwan  8400960 Feb 15 00:36 lot_20210214_0094.fits
-rw-r--r--  1 daisuke  taiwan  8400960 Feb 15 00:37 lot_20210214_0095.fits
-rw-r--r--  1 daisuke  taiwan  8400960 Feb 15 00:37 lot_20210214_0096.fits
% ls data_ao2021_s13 | wc
     534     534   12282
```

3 Data reduction

Make your own data reduction pipeline for optical imaging data using Python. Here is an example.

Python Code 1: nimccdred.py

```python
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing datetime module
import datetime

# importing numpy module
import numpy
import numpy.ma

# importing astropy module
import astropy.io.fits
import astropy.stats

##############################################################################

# date/time
now      = datetime.datetime.now ()
YYYYMMDD = "%04d%02d%02d" % (now.year, now.month, now.day)
HHMMSS   = "%02d%02d%02d" % (now.hour, now.minute, now.second)
```

```
##############################################################################

# constructing parser object
desc = "Data reduction pipeline for 2-dim optical imaging CCD data"
parser = argparse.ArgumentParser (description=desc)

# default values
default_dir     = "ccdred_%s_%s" % (YYYYMMDD, HHMMSS)
default_logfile = "%s/ccdred.log" % (default_dir)
list_cenfunc    = ['mean', 'median']

# adding arguments
parser.add_argument ('--keyword-naxis', default='NAXIS', \
                     help='FITS keyword for number of axes (default: NAXIS)')
parser.add_argument ('--keyword-naxis1', default='NAXIS1', \
                     help='FITS keyword for NAXIS1 (default: NAXIS1)')
parser.add_argument ('--keyword-naxis2', default='NAXIS2', \
                     help='FITS keyword for NAXIS2 (default: NAXIS2)')
parser.add_argument ('--keyword-exptime', default='EXPTIME', \
                     help='FITS keyword for exposure time (default: EXPTIME)')
parser.add_argument ('--keyword-filter', default='FILTER', \
                     help='FITS keyword for filter name (default: FILTER)')
parser.add_argument ('--keyword-datatype', default='IMAGETYP', \
                     help='FITS keyword for data type (default: IMAGETYP)')
parser.add_argument ('--keyword-datatype-bias', default='BIAS', \
                     help='keyword value for bias frame (default: BIAS)')
parser.add_argument ('--keyword-datatype-dark', default='DARK', \
                     help='keyword value for dark frame (default: DARK)')
parser.add_argument ('--keyword-datatype-flat', default='FLAT', \
                     help='keyword value for flat frame (default: FLAT)')
parser.add_argument ('--keyword-datatype-object', default='LIGHT', \
                     help='keyword value for object frame (default: LIGHT)')
parser.add_argument ('-l', '--logfile', default=default_logfile, \
                     help='log file name')
parser.add_argument ('-d', '--dir-reduced', default=default_dir, \
                     help='directory to store reduced data')
parser.add_argument ('-t', '--threshold', type=float, default=2.5, \
                     help='threshold for sigma-clipping (default: 2.5)')
parser.add_argument ('-c', '--cenfunc', choices=list_cenfunc, \
                     default='median', \
                     help='method to estimate centre value (default: median)')
parser.add_argument ('-i', '--maxiters', type=int, default=10, \
                     help='maximum number of iterations (default: 10)')
parser.add_argument ('-u', '--upperlimit', type=float, default=38000.0, \
                     help='upper limit of pixel value for use (default: 38000)')
parser.add_argument ('files', nargs='+', help='FITS files')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
keyword_naxis           = args.keyword_naxis
keyword_naxis1          = args.keyword_naxis1
keyword_naxis2          = args.keyword_naxis2
keyword_exptime         = args.keyword_exptime
keyword_filter          = args.keyword_filter
keyword_datatype        = args.keyword_datatype
keyword_datatype_bias   = args.keyword_datatype_bias
keyword_datatype_dark   = args.keyword_datatype_dark
```

```
keyword_datatype_flat   = args.keyword_datatype_flat
keyword_datatype_object = args.keyword_datatype_object
threshold               = args.threshold
cenfunc                 = args.cenfunc
maxiters                = args.maxiters
upper_limit_adu         = args.upperlimit
file_logfile            = args.logfile
files_fits              = args.files
dir_red                 = args.dir_reduced

##############################################################################

# functions

def make_filename_combineddark (exptime_msec):
    file_combineddark = "dark_%08d.fits" % (exptime_msec)
    return (file_combineddark)

def make_filename_combinedflatfield (filter):
    file_combinedflatfield = "flat_%s.fits" % (filter)
    return (file_combinedflatfield)

def make_filename_darksub (file_raw):
    file_darksub = file_raw.split ('/')[-1][:-5] + '_d.fits'
    return (file_darksub)

def make_filename_flatfielded (file_raw):
    file_flatfielded = file_raw.split ('/')[-1][:-5] + '_df.fits'
    return (file_flatfielded)

# function for combining dark-subtracted flat-field frames
def combine_flat (path_combined_flatfield, list_flat, \
                  threshold, cenfunc, maxiters):
    # if number of flat frames is less than 2, then stop the script.
    if ( len (list_flat) < 2 ):
        # error message for log file
        fh_log.write ("#\n")
        fh_log.write ("# ERROR: there is only one flat-field frame!\n")
        fh_log.write ("#\n")
        # exit
        sys.exit ()

    # counter
    i = 0

    # making an empty list
    list_median = []

    # reading flat-field frames
    for file_flat in sorted (list_flat):
        # opening FITS file
        hdu_list = astropy.io.fits.open (file_flat)

        # header of primary HDU (only for the first file)
        if (i == 0):
            header = hdu_list[0].header

        # image data of primary HDU (reading the data as float64)
        data = hdu_list[0].data.astype (numpy.float64)
```

```
        # closing FITS file
        hdu_list.close ()

        # median pixel value of first image
        if (i == 0):
            median_ref = numpy.median (data)

        # median pixel value
        median = numpy.median (data)

        # if median value of flat-field frame is too high, then skip
        if (median > upper_limit_adu):
            print ("#   %s: median = %d (REJECTED)" \
                   % (file_flat, median) )
            continue
        print ("#   %s: median = %d" % (file_flat, median) )

        # appending median value to the list
        list_median.append (median)

        # scaling
        data_scaled = data / median * median_ref

        # constructing data cube
        if (i == 0):
            data_tmp = data_scaled
        elif (i == 1):
            cube = numpy.concatenate ( ([data_tmp], [data_scaled]), axis=0)
        else:
            cube = numpy.concatenate ( (cube, [data_scaled]), axis=0)

        # incrementing counter "i"
        i += 1

    # combining flat-field frames
    cube_clipped = astropy.stats.sigma_clip (cube, sigma=threshold, \
                                             maxiters=maxiters, \
                                             cenfunc=cenfunc, axis=0, \
                                             masked=True)
    flatfield_combined = numpy.ma.average (cube_clipped, weights=list_median, \
                                           axis=0)

    # normalisation
    mean_flatfield = numpy.ma.average (flatfield_combined)
    flatfield_normalised = flatfield_combined / mean_flatfield

    # now
    datetime_now = datetime.datetime.now ()

    # adding comments to header
    header['comment'] = "Updated on %s" % (datetime_now)
    header['comment'] = "Multiple flat frames are combined into a FITS file"
    header['comment'] = "List of combined flat-field frames:"
    for file_flat in list_flat:
        header['comment'] = "  %s" % (file_flat)
    header['comment'] = "Options:"
    header['comment'] = "  threshold = %f sigma" % (threshold)
    header['comment'] = "  cenfunc   = %s" % (cenfunc)
```

```
    header['comment'] = "  maxiters  = %d" % (maxiters)

    # writing a new FITS file
    astropy.io.fits.writeto (path_combined_flatfield, \
                             numpy.ma.filled (flatfield_normalised, \
                                              fill_value=1.0), \
                             header=header)

    # return True
    return (True)

# function for combining dark frames
def combine_dark (path_combined_dark, list_dark, \
                  threshold, cenfunc, maxiters):
    # if number of dark frames is less than 2, then stop the script.
    if ( len (list_dark) < 2 ):
        # error message for log file
        fh_log.write ("#\n")
        fh_log.write ("# ERROR: there is only one dark frame!\n")
        fh_log.write ("#\n")
        # exit
        sys.exit ()

    # counter
    i = 0

    # reading dark frames
    for file_dark in sorted (list_dark):
        # opening FITS file
        hdu_list = astropy.io.fits.open (file_dark)

        # header of primary HDU (only for the first file)
        if (i == 0):
            header = hdu_list[0].header

        # image data of primary HDU (reading the data as float64)
        data = hdu_list[0].data.astype (numpy.float64)

        # closing FITS file
        hdu_list.close ()

        # sigma-clipping
        data_clipped = astropy.stats.sigma_clip (data, sigma=threshold, \
                                                 maxiters=maxiters, \
                                                 cenfunc=cenfunc, \
                                                 axis=None, masked=True)
        data_mask = data_clipped.mask

        # constructing data cube and its mask
        if (i == 0):
            data_tmp = data
            mask_tmp = data_mask
        elif (i == 1):
            cube      = numpy.concatenate ( ([data_tmp], [data]), axis=0)
            cube_mask = numpy.concatenate ( ([mask_tmp], [data_mask]), axis=0)
        else:
            cube      = numpy.concatenate ( (cube, [data]), axis=0)
            cube_mask = numpy.concatenate ( (cube_mask, [data_mask]), axis=0)
```

```
        # incrementing counter "i"
        i += 1

    # constructing a masked data cube
    cube_masked = numpy.ma.array (cube, mask=cube_mask)

    # combining dark frames
    cube_clipped = astropy.stats.sigma_clip (cube_masked, sigma=threshold, \
                                             maxiters=maxiters, \
                                             cenfunc=cenfunc, axis=0, \
                                             masked=True)
    dark_combined = numpy.ma.average (cube_clipped, axis=0)

    # now
    datetime_now = datetime.datetime.now ()

    # adding comments to header
    header['comment'] = "Updated on %s" % (datetime_now)
    header['comment'] = "Multiple dark frames are combined into a FITS file"
    header['comment'] = "List of combined dark frames:"
    for file_dark in list_dark:
        header['comment'] = "  %s" % (file_dark)
    header['comment'] = "Options:"
    header['comment'] = "  threshold = %f sigma" % (threshold)
    header['comment'] = "  cenfunc   = %s" % (cenfunc)
    header['comment'] = "  maxiters  = %d" % (maxiters)

    # writing a new FITS file
    mean = numpy.ma.mean (dark_combined)
    astropy.io.fits.writeto (path_combined_dark, \
                             numpy.ma.filled (dark_combined, fill_value=mean), \
                             header=header)

    # return True
    return (True)

def image_subtraction (file_1, file_2, file_result):
    # opening FITS file
    hdu_list_1 = astropy.io.fits.open (file_1)

    # reading header
    header = hdu_list_1[0].header

    # reading image data
    data1 = hdu_list_1[0].data.astype (numpy.float64)

    # closing FITS file
    hdu_list_1.close ()

    # opening FITS file
    hdu_list_2 = astropy.io.fits.open (file_2)

    # reading image data
    data2 = hdu_list_2[0].data.astype (numpy.float64)

    # closing FITS file
    hdu_list_2.close ()

    # image subtraction
```

```
    data_result = data1 - data2

    # now
    datetime_now = datetime.datetime.now ()

    # adding comments to header
    header['comment'] = "Updated on %s" % (datetime_now)
    header['comment'] = "Image subtraction was carried out"
    header['comment'] = "Operation: %s - %s" % (file_1, file_2)
    header['comment'] = "New file: %s" % (file_result)

    # writing a new FITS file
    astropy.io.fits.writeto (file_result, data_result, header=header)

    # return True
    return (True)

def image_division (file_1, file_2, file_result):
    # opening FITS file
    hdu_list_1 = astropy.io.fits.open (file_1)

    # reading header
    header = hdu_list_1[0].header

    # reading image data
    data1 = hdu_list_1[0].data.astype (numpy.float64)

    # closing FITS file
    hdu_list_1.close ()

    # opening FITS file
    hdu_list_2 = astropy.io.fits.open (file_2)

    # reading image data
    data2 = hdu_list_2[0].data.astype (numpy.float64)

    # closing FITS file
    hdu_list_2.close ()

    # image subtraction
    data_result = data1 / data2

    # now
    datetime_now = datetime.datetime.now ()

    # adding comments to header
    header['comment'] = "Updated on %s" % (datetime_now)
    header['comment'] = "Image division was carried out"
    header['comment'] = "Operation: %s / %s" % (file_1, file_2)
    header['comment'] = "New file: %s" % (file_result)

    # writing a new FITS file
    astropy.io.fits.writeto (file_result, data_result, header=header)

    # return True
    return (True)

#############################################################################
```

```
# making directories

# making directory for reduced data
path_dir_red = pathlib.Path (dir_red)
path_dir_red.mkdir (exist_ok=True)

################################################################################

# dictionary for header information
dic_header = {}

# list for exposure time
list_exptime = []

# list for filter name
list_filter = []

# list for raw object frames
list_object_raw = []

# list for dark-subtracted object frames
list_object_darksub = []

# list for flat-fielded object frames
list_object_flatfielded = []

# list for raw flat-field frames
list_flat_raw   = []

# list for dark-subtracted flat-field frames
list_flat_darksub = []

################################################################################

# opening log file for writing

fh_log = open (file_logfile, 'w')

################################################################################

# printing status
print ("# checking whether files are FITS files...")

# checking all the files
for file_fits in sorted (files_fits):
    # if the file is not a FITS file, then stop the script.
    if not (file_fits[-5:] == '.fits'):
        # error message for log file
        fh_log.write ("#\n")
        fh_log.write ("# ERROR: %s is not a FITS file!\n" % file_fits)
        fh_log.write ("#\n")
        # exit
        sys.exit ()

# printing status
print ("# checking whether files are FITS files done!")

################################################################################
```

```
# printing status
print ("# reading header information of FITS files...")

# reading FITS header information
for file_fits in sorted (files_fits):
    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # reading FITS header
    header = hdu_list[0].header

    # number of axes
    naxis = header[keyword_naxis]
    # number of pixels of x-axis
    naxis1 = header[keyword_naxis1]
    # number of pixels of y-axis
    naxis2 = header[keyword_naxis2]
    # data type
    datatype = header[keyword_datatype]
    # exposure time
    exptime = header[keyword_exptime]
    # filter name
    if ( (datatype == keyword_datatype_object) \
         or (datatype == keyword_datatype_flat) ):
        filter = header[keyword_filter]
    else:
        filter = '__NONE__'

    # closing FITS file
    hdu_list.close ()

    # adding data to dictionary for header information
    if not (file_fits in dic_header):
        dic_header[file_fits]             = {}
        dic_header[file_fits]['naxis']    = naxis
        dic_header[file_fits]['naxis1']   = naxis1
        dic_header[file_fits]['naxis2']   = naxis2
        dic_header[file_fits]['datatype'] = datatype
        dic_header[file_fits]['exptime']  = exptime
        dic_header[file_fits]['filter']   = filter

# printing status
print ("# reading header information of FITS files done!")

##############################################################################

# writing FITS header information to log file
fh_log.write ("#\n")
fh_log.write ("# Input files\n")
fh_log.write ("#\n")
for file_fits in sorted (dic_header):
    fh_log.write ("# %s\n" % (file_fits) )
    fh_log.write ("#   NAXIS    = %d\n" % (dic_header[file_fits]['naxis']) )
    fh_log.write ("#   NAXIS1   = %d\n" % (dic_header[file_fits]['naxis1']) )
    fh_log.write ("#   NAXIS2   = %d\n" % (dic_header[file_fits]['naxis2']) )
    fh_log.write ("#   datatype = %s\n" % (dic_header[file_fits]['datatype']) )
    fh_log.write ("#   exptime  = %f\n" % (dic_header[file_fits]['exptime']) )
    fh_log.write ("#   filter   = %s\n" % (dic_header[file_fits]['filter']) )
fh_log.write ("#\n")
```

```
fh_log.write ("#\n")
fh_log.write ("#\n")

##############################################################################

# printing status
print ("# checking image dimensions...")

# check of number of axes
for file_fits in sorted (dic_header):
    # if the FITS file is not 2-dim image, then stop the script.
    if (dic_header[file_fits]['naxis'] != 2):
        # error message for log file
        fh_log.write ("#\n")
        fh_log.write ("# ERROR: number of axes of %s is not 2!\n" % file_fits)
        fh_log.write ("#\n")
        # exit
        sys.exit ()

# printing status
print ("# checking image dimensions done!")

##############################################################################

# printing status
print ("# checking image size...")

# counter
i = 0
# check of image size
for file_fits in sorted (dic_header):
    # image size of reference data
    if (i == 0):
        # number of pixels in x-axis of ref. image
        image_size_x = dic_header[file_fits]['naxis1']
        # number of pixels in y-axis of ref. image
        image_size_y = dic_header[file_fits]['naxis2']

    # check of image size
    if ( (dic_header[file_fits]['naxis1'] != image_size_x) \
         or (dic_header[file_fits]['naxis2'] != image_size_y) ):
        # error message for log file
        fh_log.write ("#\n")
        fh_log.write ("# ERROR: size of %s is different from ref. image!\n" \
                      % file_fits)
        fh_log.write ("# ERROR: image size of ref. data = (%d, %d)\n" \
                      % (image_size_x, image_size_y) )
        fh_log.write ("# ERROR: image size of %s = (%d, %d)\n" \
                      % (file_fits, dic_header[file_fits]['naxis1'], \
                         dic_header[file_fits]['naxis2']) )
        fh_log.write ("#\n")
        # exit
        sys.exit ()

    # incrementing counter
    i += 1

# printing status
print ("# checking image size done!")
```

```
################################################################################

# printing status
print ("# identifying necessary dark and flat-field...")

# making a list of exposure time and filter name
for file_fits in sorted (dic_header):
    if ( (dic_header[file_fits]['datatype'] == keyword_datatype_object) \
         or (dic_header[file_fits]['datatype'] == keyword_datatype_flat) ):
        # adding exposure time to the list
        if not (dic_header[file_fits]['exptime'] in list_exptime):
            list_exptime.append (dic_header[file_fits]['exptime'])
        if (dic_header[file_fits]['datatype'] == keyword_datatype_object):
            # adding filter name to the list
            if not (dic_header[file_fits]['filter'] in list_filter):
                list_filter.append (dic_header[file_fits]['filter'])

# output for log file
fh_log.write ("#\n")
fh_log.write ("# exposure time and filters\n")
fh_log.write ("#\n")
fh_log.write ("#   exposure time\n")
for exptime in sorted (list_exptime):
    fh_log.write ("#     %f sec\n" % exptime)
fh_log.write ("#   filters\n")
for filter in sorted (list_filter):
    fh_log.write ("#     %s\n" % filter)
fh_log.write ("#\n")
fh_log.write ("#\n")
fh_log.write ("#\n")

# printing status
print ("# identifying necessary dark and flat-field done!")

################################################################################

# printing status
print ("# checking dark and flat-field availability...")

# check of dark
for exptime in sorted (list_exptime):
    has_dark = 'NO'
    for file_fits in sorted (dic_header):
        # if not dark, then skip
        if not (dic_header[file_fits]['datatype'] == keyword_datatype_dark):
            continue
        # if we find dark for given exptime, then set 'has_dark'
        if (dic_header[file_fits]['exptime'] == exptime):
            has_dark = 'YES'
    # if there is no dark for given exptime, then stop the script
    if (has_dark == 'NO'):
        fh_log.write ("#\n")
        fh_log.write ("# ERROR: no dark for %f sec!\n" % exptime)
        fh_log.write ("#\n")
        sys.exit ()

# check of flat-field
for filter in sorted (list_filter):
```

```
    has_flat = 'NO'
    for file_fits in sorted (dic_header):
        # if not flat-field, then skip
        if not (dic_header[file_fits]['datatype'] == keyword_datatype_flat):
            continue
        # if we find flat-field for given filter name, then set 'has_flat'
        if (dic_header[file_fits]['filter'] == filter):
            has_flat = 'YES'
    # if there is no flat-field for given filter, then stop the script
    if (has_flat == 'NO'):
        fh_log.write ("#\n")
        fh_log.write ("# ERROR: no flatfield for %s band!\n" % filter)
        fh_log.write ("#\n")
        sys.exit ()

# printing status
print ("# checking dark and flat-field availability done!")

################################################################################

# classifying data

# searching for all the files
for file_fits in sorted (dic_header):
    # if object frame
    if (dic_header[file_fits]['datatype'] == keyword_datatype_object):
        # appending file name to the list of raw object frames
        list_object_raw.append (file_fits)
    # if flat-field frame
    if (dic_header[file_fits]['datatype'] == keyword_datatype_flat):
        # appending file name to the list of raw flat-field frames
        list_flat_raw.append (file_fits)

################################################################################

# combining dark frames

# message for log file
fh_log.write ("#\n")
fh_log.write ("# combining dark frames\n")
fh_log.write ("#\n")

# printing status
print ("# combining dark frames...")

# for each exposure time, combine dark frames
for exptime in sorted (list_exptime):
    # list of dark frames
    list_dark = []
    # exposure time in msec
    exptime_msec = exptime * 1000
    # combined dark frame file name
    file_combined_dark = make_filename_combineddark (exptime_msec)
    path_combined_dark = "%s/%s" % (dir_red, file_combined_dark)
    # searching dark frames of given exposure time
    for file_fits in sorted (dic_header):
        # if not dark frame, then skip
        if not (dic_header[file_fits]['datatype'] == keyword_datatype_dark):
            continue
```

```
        # appending file name to the list, if exposure time matches
        if (dic_header[file_fits]['exptime'] == exptime):
            list_dark.append (file_fits)

    # printing status
    print ("#  making %f sec combined dark frame..." % exptime)

    # combine dark frames
    combine_dark (path_combined_dark, list_dark, \
                  threshold, cenfunc, maxiters)

    # message for log file
    fh_log.write ("#   making combined dark frame %s\n" % path_combined_dark)
    for file_dark in sorted (list_dark):
        fh_log.write ("#     %s\n" % file_dark)

    # printing status
    print ("#  making %f sec combined dark frame done!" % exptime)

# printing status
print ("# combining dark frames done!")

# message for log file
fh_log.write ("#\n")
fh_log.write ("#\n")
fh_log.write ("#\n")

################################################################################

# dark subtraction for flat-field frames

# message for log file
fh_log.write ("#\n")
fh_log.write ("# dark subtraction for flat-field frames\n")
fh_log.write ("#\n")

# printing status
print ("# subtracting combined dark from flat-field frames...")

# subtracting dark frame from raw flat-field frame
for file_fits in sorted (list_flat_raw):
    # file name of dark subtracted FITS file
    file_darksub = make_filename_darksub (file_fits)
    # path name of dark subtracted FITS file
    path_darksub = "%s/%s" % (dir_red, file_darksub)
    # appending path name of dark subtracted FITS file to the list
    list_flat_darksub.append (path_darksub)

    # exposure time in msec
    exptime_msec = dic_header[file_fits]['exptime'] * 1000
    # file name of combined dark frame
    file_combineddark = make_filename_combineddark (exptime_msec)
    path_combineddark = "%s/%s" % (dir_red, file_combineddark)

    # printing status
    print ("#  %s" % (file_fits) )
    print ("#   - %s" % (path_combineddark) )
    print ("#    ==> %s" % (path_darksub) )
```

```
    # message for log file
    fh_log.write ("#  %s\n" % (file_fits) )
    fh_log.write ("#    - %s\n" % (path_combineddark) )
    fh_log.write ("#     ==> %s\n" % (path_darksub) )

    # image subtraction
    image_subtraction (file_fits, path_combineddark, path_darksub)

# printing status
print ("# subtracting combined dark from flat-field frames done!")

# message for log file
fh_log.write ("#\n")
fh_log.write ("#\n")
fh_log.write ("#\n")

# printing status
print ("# combining flat-field...")

# message for log file
fh_log.write ("#\n")
fh_log.write ("# combining flat-field\n")
fh_log.write ("#\n")

# combining dark-subtracted flat-field frames
for filter in list_filter:
    # list of files to be combined
    list_combine = []
    for file_fits in sorted (list_flat_darksub):
        # opening FITS file
        hdu_list = astropy.io.fits.open (file_fits)
        # reading header information
        header = hdu_list[0].header
        # closing FITS file
        hdu_list.close ()
        # appending file name to the list
        if (header[keyword_filter] == filter):
            list_combine.append (file_fits)

    # file name of combined flat-field frame
    file_combined_flatfield = make_filename_combinedflatfield (filter)
    path_combined_flatfield = "%s/%s" % (dir_red, file_combined_flatfield)

    # printing status
    print ("#  making %s band flatfield..." % (filter) )

    # combine dark-subtracted flat-field frames
    combine_flat (path_combined_flatfield, list_combine, \
                  threshold, cenfunc, maxiters)

    # printing status
    print ("#  making %s band flatfield done!" % (filter) )

    # message for log file
    fh_log.write ("#\n")
    fh_log.write ("# combining flat-field %s\n" % path_combined_flatfield)
    for file_fits in list_combine:
        fh_log.write ("#   %s\n" % file_fits)
    fh_log.write ("#\n")
```

```
# printing status
print ("# combining flat-field done!")

# message for log file
fh_log.write ("#\n")
fh_log.write ("#\n")
fh_log.write ("#\n")

##############################################################################

# printing status
print ("# processing object frames...")

# message for log file
fh_log.write ("#\n")
fh_log.write ("# processing object frames\n")
fh_log.write ("#\n")

# subtracting dark frame from raw object frame
for file_fits in sorted (list_object_raw):
    # file name of dark subtracted FITS file
    file_darksub = make_filename_darksub (file_fits)
    # path name of dark subtracted FITS file
    path_darksub = "%s/%s" % (dir_red, file_darksub)

    # file name of flat-fielded FITS file
    file_flatfielded = make_filename_flatfielded (file_fits)
    # path name of flat-fielded FITS file
    path_flatfielded = "%s/%s" % (dir_red, file_flatfielded)

    # exposure time in msec
    exptime_msec = dic_header[file_fits]['exptime'] * 1000
    # file name of combined dark frame
    file_combineddark = make_filename_combineddark (exptime_msec)
    path_combineddark = "%s/%s" % (dir_red, file_combineddark)

    # filter name
    filter = dic_header[file_fits]['filter']
    # file name of combined flat-field frame
    file_combinedflatfield = make_filename_combinedflatfield (filter)
    path_combinedflatfield = "%s/%s" % (dir_red, file_combinedflatfield)

    # printing status
    print ("#  dark subtraction of %s" % file_fits)
    print ("#   %s" % (file_fits) )
    print ("#    - %s" % (path_combineddark) )
    print ("#     ==> %s" % (path_darksub) )

    # message for log file
    fh_log.write ("#  dark subtraction of %s\n" % file_fits)
    fh_log.write ("#   %s\n" % (file_fits) )
    fh_log.write ("#    - %s\n" % (path_combineddark) )
    fh_log.write ("#     ==> %s\n" % (path_darksub) )

    # image subtraction
    image_subtraction (file_fits, path_combineddark, path_darksub)

    # printing status
```

```
    print ("#  flatfielding of %s" % path_darksub)
    print ("#   %s" % (path_darksub) )
    print ("#    / %s" % (path_combinedflatfield) )
    print ("#     ==> %s" % (path_flatfielded) )

    # printing status
    fh_log.write ("#  flatfielding of %s\n" % path_darksub)
    fh_log.write ("#   %s\n" % (path_darksub) )
    fh_log.write ("#    / %s\n" % (path_combinedflatfield) )
    fh_log.write ("#     ==> %s\n" % (path_flatfielded) )

    # image division
    image_division (path_darksub, path_combinedflatfield, path_flatfielded)

# printing status
print ("# processing object frames done!")

# message for log file
fh_log.write ("#\n")
fh_log.write ("#\n")
fh_log.write ("#\n")

################################################################################

# closing log file
fh_log.close ()
```

Execute the pipeline.

```
% ./nimccdred.py data_ao2021_s13/*.fits
# checking whether files are FITS files...
# checking whether files are FITS files done!
# reading header information of FITS files...
# reading header information of FITS files done!
# checking image dimensions...
# checking image dimensions done!
# checking image size...
# checking image size done!
# identifying necessary dark and flat-field...
# identifying necessary dark and flat-field done!
# checking dark and flat-field availability...
# checking dark and flat-field availability done!
# combining dark frames...
#  making 5.000000 sec combined dark frame...
#  making 5.000000 sec combined dark frame done!
#  making 10.000000 sec combined dark frame...
#  making 10.000000 sec combined dark frame done!
#  making 15.000000 sec combined dark frame...
#  making 15.000000 sec combined dark frame done!
#  making 45.000000 sec combined dark frame...
#  making 45.000000 sec combined dark frame done!
#  making 60.000000 sec combined dark frame...
#  making 60.000000 sec combined dark frame done!
#  making 180.000000 sec combined dark frame...
#  making 180.000000 sec combined dark frame done!
# combining dark frames done!
# subtracting combined dark from flat-field frames...
#  data_ao2021_s13/lot_20210214_0352.fits
```

```
#    - ccdred_20210507_014047/dark_00060000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0352_d.fits
#   data_ao2021_s13/lot_20210214_0353.fits
#    - ccdred_20210507_014047/dark_00060000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0353_d.fits
#   data_ao2021_s13/lot_20210214_0354.fits
#    - ccdred_20210507_014047/dark_00060000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0354_d.fits

.....

#   data_ao2021_s13/lot_20210214_0447.fits
#    - ccdred_20210507_014047/dark_00005000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0447_d.fits
#   data_ao2021_s13/lot_20210214_0448.fits
#    - ccdred_20210507_014047/dark_00005000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0448_d.fits
#   data_ao2021_s13/lot_20210214_0449.fits
#    - ccdred_20210507_014047/dark_00005000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0449_d.fits
#   data_ao2021_s13/lot_20210214_0450.fits
#    - ccdred_20210507_014047/dark_00005000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0450_d.fits
#   data_ao2021_s13/lot_20210214_0451.fits
#    - ccdred_20210507_014047/dark_00005000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0451_d.fits
# subtracting combined dark from flat-field frames done!
# combining flat-field...
#  making rp_Astrodon_2019 band flatfield...
#   ccdred_20210507_014047/lot_20210214_0396_d.fits: median = 269
#   ccdred_20210507_014047/lot_20210214_0401_d.fits: median = 636
#   ccdred_20210507_014047/lot_20210214_0406_d.fits: median = 1725
#   ccdred_20210507_014047/lot_20210214_0411_d.fits: median = 4561
#   ccdred_20210507_014047/lot_20210214_0416_d.fits: median = 11531
#   ccdred_20210507_014047/lot_20210214_0422_d.fits: median = 10367
#   ccdred_20210507_014047/lot_20210214_0427_d.fits: median = 18252
#   ccdred_20210507_014047/lot_20210214_0432_d.fits: median = 11087
#   ccdred_20210507_014047/lot_20210214_0437_d.fits: median = 16149
#   ccdred_20210507_014047/lot_20210214_0442_d.fits: median = 24107
#   ccdred_20210507_014047/lot_20210214_0447_d.fits: median = 37051
#  making rp_Astrodon_2019 band flatfield done!
#  making gp_Astrodon_2019 band flatfield...
#   ccdred_20210507_014047/lot_20210214_0394_d.fits: median = 229
#   ccdred_20210507_014047/lot_20210214_0399_d.fits: median = 539
#   ccdred_20210507_014047/lot_20210214_0404_d.fits: median = 1340
#   ccdred_20210507_014047/lot_20210214_0409_d.fits: median = 3394
#   ccdred_20210507_014047/lot_20210214_0414_d.fits: median = 8783
#   ccdred_20210507_014047/lot_20210214_0419_d.fits: median = 24869
#   ccdred_20210507_014047/lot_20210214_0420_d.fits: median = 10849
#   ccdred_20210507_014047/lot_20210214_0425_d.fits: median = 19192
#   ccdred_20210507_014047/lot_20210214_0430_d.fits: median = 12579
#   ccdred_20210507_014047/lot_20210214_0435_d.fits: median = 17903
#   ccdred_20210507_014047/lot_20210214_0440_d.fits: median = 25604
#   ccdred_20210507_014047/lot_20210214_0445_d.fits: median = 36497
#  making gp_Astrodon_2019 band flatfield done!
#  making ip_Astrodon_2019 band flatfield...
#   ccdred_20210507_014047/lot_20210214_0352_d.fits: median = 102
#   ccdred_20210507_014047/lot_20210214_0353_d.fits: median = 101
#   ccdred_20210507_014047/lot_20210214_0354_d.fits: median = 101
```

```
#   ccdred_20210507_014047/lot_20210214_0398_d.fits: median = 370
#   ccdred_20210507_014047/lot_20210214_0403_d.fits: median = 974
#   ccdred_20210507_014047/lot_20210214_0408_d.fits: median = 2706
#   ccdred_20210507_014047/lot_20210214_0413_d.fits: median = 6508
#   ccdred_20210507_014047/lot_20210214_0418_d.fits: median = 14297
#   ccdred_20210507_014047/lot_20210214_0424_d.fits: median = 9764
#   ccdred_20210507_014047/lot_20210214_0429_d.fits: median = 17453
#   ccdred_20210507_014047/lot_20210214_0434_d.fits: median = 9747
#   ccdred_20210507_014047/lot_20210214_0439_d.fits: median = 14698
#   ccdred_20210507_014047/lot_20210214_0444_d.fits: median = 23373
#   ccdred_20210507_014047/lot_20210214_0449_d.fits: median = 39062 (REJECTED)
#  making ip_Astrodon_2019 band flatfield done!
# combining flat-field done!
# processing object frames...
#  dark subtraction of data_ao2021_s13/lot_20210214_0085.fits
#   data_ao2021_s13/lot_20210214_0085.fits
#    - ccdred_20210507_014047/dark_00060000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0085_d.fits
#  flatfielding of ccdred_20210507_014047/lot_20210214_0085_d.fits
#   ccdred_20210507_014047/lot_20210214_0085_d.fits
#    / ccdred_20210507_014047/flat_rp_Astrodon_2019.fits
#     ==> ccdred_20210507_014047/lot_20210214_0085_df.fits
#  dark subtraction of data_ao2021_s13/lot_20210214_0086.fits
#   data_ao2021_s13/lot_20210214_0086.fits
#    - ccdred_20210507_014047/dark_00060000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0086_d.fits
#  flatfielding of ccdred_20210507_014047/lot_20210214_0086_d.fits
#   ccdred_20210507_014047/lot_20210214_0086_d.fits
#    / ccdred_20210507_014047/flat_rp_Astrodon_2019.fits
#     ==> ccdred_20210507_014047/lot_20210214_0086_df.fits

.....

#  dark subtraction of data_ao2021_s13/lot_20210214_0388.fits
#   data_ao2021_s13/lot_20210214_0388.fits
#    - ccdred_20210507_014047/dark_00060000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0388_d.fits
#  flatfielding of ccdred_20210507_014047/lot_20210214_0388_d.fits
#   ccdred_20210507_014047/lot_20210214_0388_d.fits
#    / ccdred_20210507_014047/flat_rp_Astrodon_2019.fits
#     ==> ccdred_20210507_014047/lot_20210214_0388_df.fits
#  dark subtraction of data_ao2021_s13/lot_20210214_0389.fits
#   data_ao2021_s13/lot_20210214_0389.fits
#    - ccdred_20210507_014047/dark_00060000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0389_d.fits
#  flatfielding of ccdred_20210507_014047/lot_20210214_0389_d.fits
#   ccdred_20210507_014047/lot_20210214_0389_d.fits
#    / ccdred_20210507_014047/flat_rp_Astrodon_2019.fits
#     ==> ccdred_20210507_014047/lot_20210214_0389_df.fits
#  dark subtraction of data_ao2021_s13/lot_20210214_0390.fits
#   data_ao2021_s13/lot_20210214_0390.fits
#    - ccdred_20210507_014047/dark_00060000.fits
#     ==> ccdred_20210507_014047/lot_20210214_0390_d.fits
#  flatfielding of ccdred_20210507_014047/lot_20210214_0390_d.fits
#   ccdred_20210507_014047/lot_20210214_0390_d.fits
#    / ccdred_20210507_014047/flat_rp_Astrodon_2019.fits
#     ==> ccdred_20210507_014047/lot_20210214_0390_df.fits
# processing object frames done!
```

4 For your training

1. Read chapter 4 of "Handbook of CCD Astronomy" to learn about basic CCD data reduction.
   - Handbook of CCD Astronomy (2nd Edition)
     - Steve B. Howell
     - Cambridge University Press
     - `https://doi.org/10.1017/CBO9780511807909`
2. Read the document "A User's Guide to CCD Reductions with IRAF" and learn about basic CCD data reduction.
   - A User's Guide to CCD Reductions with IRAF
     - Philip Massey
     - `http://iraf.noao.edu/iraf/ftp/docs/ccduser3.ps.Z`
3. Read the document "An Introduction to Astronomical Photometry using CCDs" and learn about basic CCD data reduction.
   - An Introduction to Astronomical Photometry using CCDs
     - William Romanishin
     - `http://hildaandtrojanasteroids.net/wrccd22oct06.pdf`

5 Assignment

No assignment for this session.