Advanced Astronomical Observations 2021
Session 08: Basic CCD Data Reduction

Kinoshita Daisuke

07 April 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/`

For this session, we try basic CCD data reduction.

1 Downloading data

A set of FITS files for this session is placed at following location. Download the file. The size of the file is about 1420 MB.

- `https://s3b.astro.ncu.edu.tw/advobs_202102/data/data_ao2021_s08.tar.xz`

If you prefer to use a command-line tool, such as `curl`, then try following command.

```
% curl -k -o data_ao2021_s08.tar.xz \
? https://s3b.astro.ncu.edu.tw/advobs_202102/data/data_ao2021_s08.tar.xz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1179M  100 1179M    0     0  1689k      0  0:11:54  0:11:54 --:--:-- 4407k
% ls -l data_ao2021_s08.tar.xz
-rw-r--r--  1 daisuke  taiwan  1237046180 Apr  2 21:59 data_ao2021_s08.tar.xz
% file data_ao2021_s08.tar.xz
data_ao2021_s08.tar.xz: XZ compressed data
```

If you prefer to use a web browser, such as Firefox, then start a web browser and download the file.

2 Extracting data

The file you have downloaded is a compressed TAR archive file. To extract FITS files, try following command. 354 FITS files should be extracted from the archive file.

```
% tar xJvf data_ao2021_s08.tar.xz
x data_ao2021_s08/
x data_ao2021_s08/lot_20210215_0070.fits
x data_ao2021_s08/lot_20210215_0071.fits
x data_ao2021_s08/lot_20210215_0075.fits
x data_ao2021_s08/lot_20210215_0076.fits
x data_ao2021_s08/lot_20210215_0077.fits

.....

x data_ao2021_s08/lot_20210215_0677.fits
x data_ao2021_s08/lot_20210215_0678.fits
x data_ao2021_s08/lot_20210215_0679.fits
x data_ao2021_s08/lot_20210215_0680.fits
x data_ao2021_s08/lot_20210215_0681.fits
% ls data_ao2021_s08/*.fits | wc
     354     354   13806
```

If above command does not work on your computer, then try following.

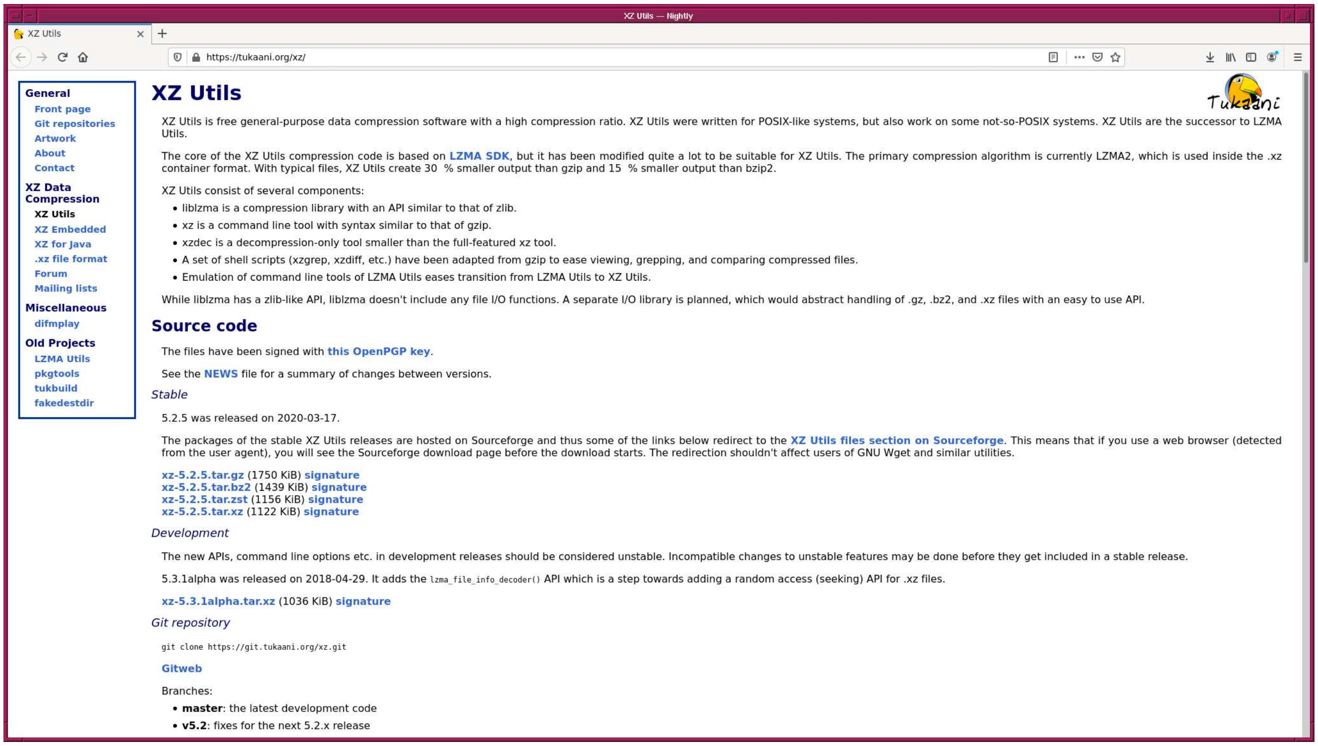
```
% unxz -c data_ao2021_s08.tar.xz | tar xvf -
x data_ao2021_s08/
x data_ao2021_s08/lot_20210215_0070.fits
x data_ao2021_s08/lot_20210215_0071.fits
x data_ao2021_s08/lot_20210215_0075.fits
x data_ao2021_s08/lot_20210215_0076.fits
x data_ao2021_s08/lot_20210215_0077.fits

.....

x data_ao2021_s08/lot_20210215_0677.fits
x data_ao2021_s08/lot_20210215_0678.fits
x data_ao2021_s08/lot_20210215_0679.fits
x data_ao2021_s08/lot_20210215_0680.fits
x data_ao2021_s08/lot_20210215_0681.fits
% ls data_ao2021_s08/*.fits | wc
     354     354   13806
```

If above command fails, you probably do not have XZ Utils. If you do not have XZ Utils, visit following website (Fig. 1) and install XZ Utils.

- `https://tukaani.org/xz/`

If you are not familiar to pipes of Unix shells, try following.

```
% ls -l data_ao2021_s08.tar.xz
-rw-r--r--  1 daisuke  taiwan  1237046180 Apr  2 22:05 data_ao2021_s08.tar.xz
% unxz data_ao2021_s08.tar.xz
% ls -l data_ao2021_s08.tar
-rw-r--r--  1 daisuke  taiwan  2972769792 Apr  2 22:05 data_ao2021_s08.tar
% tar xvf data_ao2021_s08.tar
x data_ao2021_s08/
```



XZ Utils

XZ Utils is free general-purpose data compression software with a high compression ratio. XZ Utils were written for POSIX-like systems, but also work on some not-so-POSIX systems. XZ Utils are the successor to LZMA Utils.

The core of the XZ Utils compression code is based on **LZMA SDK**, but it has been modified quite a lot to be suitable for XZ Utils. The primary compression algorithm is currently LZMA2, which is used inside the .xz container format. With typical files, XZ Utils create 30 % smaller output than gzip and 15 % smaller output than bzip2.

XZ Utils consist of several components:

- liblzma is a compression library with an API similar to that of zlib.
- xz is a command line tool with syntax similar to that of gzip.
- xzdec is a decompression-only tool smaller than the full-featured xz tool.
- A set of shell scripts (xzgrep, xzdiff, etc.) have been adapted from gzip to ease viewing, grepping, and comparing compressed files.
- Emulation of command line tools of LZMA Utils eases transition from LZMA Utils to XZ Utils.

While liblzma has a zlib-like API, liblzma doesn't include any file I/O functions. A separate I/O library is planned, which would abstract handling of .gz, .bz2, and .xz files with an easy to use API.

Source code

The files have been signed with **this OpenPGP key**.

See the **NEWS** file for a summary of changes between versions.

Stable

5.2.5 was released on 2020-03-17.

The packages of the stable XZ Utils releases are hosted on Sourceforge and thus some of the links below redirect to the **XZ Utils files section on Sourceforge**. This means that if you use a web browser (detected from the user agent), you will see the Sourceforge download page before the download starts. The redirection shouldn't affect users of GNU Wget and similar utilities.

xz-5.2.5.tar.gz (1750 KiB) **signature**
xz-5.2.5.tar.bz2 (1439 KiB) **signature**
xz-5.2.5.tar.zst (1156 KiB) **signature**
xz-5.2.5.tar.xz (1122 KiB) **signature**

Development

The new APIs, command line options etc. in development releases should be considered unstable. Incompatible changes to unstable features may be done before they get included in a stable release.

5.3.1alpha was released on 2018-04-29. It adds the `lzma_file_info_decoder()` API which is a step towards adding a random access (seeking) API for .xz files.

xz-5.3.1alpha.tar.xz (1036 KiB) **signature**

Git repository

```
git clone https://git.tukaani.org/xz.git
```

Gitweb

Branches:

- **master**: the latest development code
- **v5.2**: fixes for the next 5.2.x release

Figure 1: The website of XZ Utils.

```
x data_ao2021_s08/lot_20210215_0070.fits
x data_ao2021_s08/lot_20210215_0071.fits
x data_ao2021_s08/lot_20210215_0075.fits
x data_ao2021_s08/lot_20210215_0076.fits
x data_ao2021_s08/lot_20210215_0077.fits

.....

x data_ao2021_s08/lot_20210215_0677.fits
x data_ao2021_s08/lot_20210215_0678.fits
x data_ao2021_s08/lot_20210215_0679.fits
x data_ao2021_s08/lot_20210215_0680.fits
x data_ao2021_s08/lot_20210215_0681.fits
% ls data_ao2021_s08/*.fits | wc
     354     354   13806
```

3 Checking data

Examine whether or not we have self-consistent data set before starting CCD data reduction.

3.1 Generating a list of FITS files

Make a Python script to scan header part of all the FITS files and print summary information.

Python Code 1: ao2021_s08_01.py

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

# importing argparse module
import argparse
```

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

# construction of parser object
desc = 'Generating a list of FITS files'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
default_keyword = 'TIME-OBS,IMAGETYP,EXPTIME,FILTER,OBJECT'
parser.add_argument ('-k', '--keyword', default=default_keyword, \
                     help='a list of keyword to check')
parser.add_argument ('files', nargs='+', help='FITS files')

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

# input parameters
keyword = args.keyword
files   = args.files

# a list of FITS keywords
list_keyword = keyword.split (',')

# processing FITS files
for fits in files:
    # if the extension of the file is not '.fits', the we skip
    if not (fits[-5:] == '.fits'):
        continue

    # file name
    # for example, file = '/some/where/in/the/disk/abc0123.fits'
    # then, path ==> ['some', 'where', 'in', 'the', 'disk', 'abc0123.fits']
    # filename ==> 'abc0123.fits'
    path_fits = fits.split ('/')
    filename = path_fits[-1]

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

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # gathering information from FITS header
    record = "%-24s" % filename
    for key in list_keyword:
        # obtaining a value for given keyword
        if key in header:
            value = str (header[key])
        else:
            value = "__NONE__"
        # appending the value to the string "record"
        if (key == 'DATE-OBS'):
            record += "  %-10s" % value
        elif (key == 'TIME-OBS'):
            record += "  %-8s" % value
        elif (key == 'IMAGETYP'):
            record += "  %-5s" % value
        elif (key == 'EXPTIME'):
            record += "  %6.1f" % float (value)
        elif (key == 'FILTER'):
            record += "  %-16s" % value
        else:
            record += "  %s" % value

    # printing information
    print (record)
```

Execute the script, and examine which data you have.

```
% chmod a+x ao2021_s08_01.py
% ./ao2021_s08_01.py -h
usage: ao2021_s08_01.py [-h] [-k KEYWORD] files [files ...]

Generating a list of FITS files

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -k KEYWORD, --keyword KEYWORD
                        a list of keyword to check

% ./ao2021_s08_01.py data_ao2021_s08/*.fits
lot_20210215_0070.fits    16:06:00  LIGHT    60.0  rp_Astrodon_2019  V0678VIR
lot_20210215_0071.fits    16:10:40  LIGHT    60.0  rp_Astrodon_2019  V0678VIR
lot_20210215_0075.fits    16:14:57  LIGHT    60.0  rp_Astrodon_2019  V0678VIR
lot_20210215_0076.fits    16:16:12  LIGHT    60.0  rp_Astrodon_2019  V0678VIR
lot_20210215_0077.fits    16:17:26  LIGHT    60.0  rp_Astrodon_2019  V0678VIR
lot_20210215_0078.fits    16:18:40  LIGHT    60.0  rp_Astrodon_2019  V0678VIR
lot_20210215_0079.fits    16:19:54  LIGHT    60.0  rp_Astrodon_2019  V0678VIR
lot_20210215_0080.fits    16:21:09  LIGHT    60.0  rp_Astrodon_2019  V0678VIR
lot_20210215_0081.fits    16:22:24  LIGHT    60.0  rp_Astrodon_2019  V0678VIR
lot_20210215_0082.fits    16:23:38  LIGHT    60.0  rp_Astrodon_2019  V0678VIR

.....

lot_20210215_0672.fits    22:39:59  DARK    180.0  __NONE__          domeflat
lot_20210215_0673.fits    22:43:04  DARK    180.0  __NONE__          domeflat
lot_20210215_0674.fits    22:46:09  DARK    180.0  __NONE__          domeflat
lot_20210215_0675.fits    22:49:14  DARK    180.0  __NONE__          domeflat
lot_20210215_0676.fits    22:52:19  DARK    180.0  __NONE__          domeflat
lot_20210215_0677.fits    22:55:24  DARK    180.0  __NONE__          domeflat
lot_20210215_0678.fits    22:58:29  DARK    180.0  __NONE__          domeflat
lot_20210215_0679.fits    23:01:34  DARK    180.0  __NONE__          domeflat
lot_20210215_0680.fits    23:04:39  DARK    180.0  __NONE__          domeflat
lot_20210215_0681.fits    23:07:44  DARK    180.0  __NONE__          domeflat
```

We find files of data types “`LIGHT`”, “`FLAT`”, and “`DARK`”.

3.2 Checking filters used for the observation

Make a Python script to check which filters are used for object frames (or `LIGHT` frames).

Python Code 2: ao2021_s08_02.py

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

# importing argparse module
import argparse

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = ’Checking filters used for the observation’
parser = argparse.ArgumentParser (description=desc)

# adding arguments
default_filter_keyword = ’FILTER’
default_datatype_keyword = ’IMAGETYP’
parser.add_argument (’-f’, ’--filter’, default=default_filter_keyword, \
                     help=’FITS keyword for filter name’)
parser.add_argument (’-t’, ’--type’, default=default_datatype_keyword, \
                     help=’FITS keyword for data type’)
parser.add_argument (’files’, nargs=’+’, help=’FITS files’)

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

# input parameters
filter_keyword   = args.filter
datatype_keyword = args.type
files            = args.files

# declaring a dictionary for filter names
dict_filters = {}

# processing FITS files
for fits in files:
    # if the extension of the file is not ’.fits’, the we skip
    if not (fits[-5:] == ’.fits’):
        continue

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

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[datatype_keyword]

    # if the data type is not "LIGHT", the we skip the file
    if not (datatype == ’LIGHT’):
        continue

    # filter
    filter_name = header[filter_keyword]

    # if the filter name is not in the dictionary "dict_filters", then append
    # if the filter name is in the dictionary "dict_filters", then add 1
    if not (filter_name in dict_filters):
        # appending "filter_name" to the dictionary "dict_filters"
        dict_filters[filter_name] = 1
    else:
        # add 1
        dict_filters[filter_name] += 1

# printing filter list
print ("List of filters used for acquiring object frames:")
for filter_name in sorted (dict_filters.keys () ):
    print ("  %s (%d files)" % (filter_name, dict_filters[filter_name]) )
```

Run the script, and show the list of filters used.

```
% chmod a+x ao2021_s08_02.py
% ./ao2021_s08_02.py -h
usage: ao2021_s08_02.py [-h] [-f FILTER] [-t TYPE] files [files ...]

Checking filters used for the observation

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -f FILTER, --filter FILTER
                        FITS keyword for filter name
  -t TYPE, --type TYPE  FITS keyword for data type

% ./ao2021_s08_02.py data_ao2021_s08/*.fits
List of filters used for acquiring object frames:
  gp_Astrodon_2019 (19 files)
  ip_Astrodon_2019 (19 files)
  rp_Astrodon_2019 (126 files)
```

Now, we know that three filters, SDSS g', r', and i' filters were used to acquire the data.

3.3 Checking whether we have flatfield

If r'-band object frames exist, then r'-band flatfield must exist. Make a Python script to check whether we have flatfield for all the filters used to acquire object frames.

Python Code 3: ao2021_s08_03.py

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

# importing argparse module
import argparse

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'Checking filters used for the observation'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
```

```
default_filter_keyword = ’FILTER’
default_datatype_keyword = ’IMAGETYP’
parser.add_argument (’-f’, ’--filter’, default=default_filter_keyword, \
                     help=’FITS keyword for filter name’)
parser.add_argument (’-t’, ’--type’, default=default_datatype_keyword, \
                     help=’FITS keyword for data type’)
parser.add_argument (’files’, nargs=’+’, help=’FITS files’)

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

# input parameters
filter_keyword   = args.filter
datatype_keyword = args.type
files            = args.files

# declaring a dictionary for filter names
dict_filters_object = {}
dict_filters_flat   = {}

# processing FITS files
for fits in files:
    # if the extension of the file is not ’.fits’, the we skip
    if not (fits[-5:] == ’.fits’):
        continue

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

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[datatype_keyword]

    # if the data type is not "LIGHT" or "FLAT", the we skip the file
    if not ( (datatype == ’LIGHT’) or (datatype == ’FLAT’) ):
        continue

    # filter
    filter_name = header[filter_keyword]

    # if the filter name is not in the dictionary, then append
    # if the filter name is in the dictionary, then add 1
    if (datatype == ’LIGHT’):
        # object frames
        if not (filter_name in dict_filters_object):
            # appending "filter_name" to the dictionary "dict_filters_object"
            dict_filters_object[filter_name] = 1
        else:
            # add 1
            dict_filters_object[filter_name] += 1
    elif (datatype == ’FLAT’):
        # flatfield frames
        if not (filter_name in dict_filters_flat):
            # appending "filter_name" to the dictionary "dict_filters_flat"
            dict_filters_flat[filter_name] = 1
        else:
            # add 1
            dict_filters_flat[filter_name] += 1

# completeness parameter
complete = 1

# printing filter list
print ("List of filters used for acquiring object frames:")
for filter_name in sorted (dict_filters_object.keys () ):
    print ("  %s (%d files)" % (filter_name, dict_filters_object[filter_name]) )
    print ("    Do we have flatfield for %s band filter?" % (filter_name) )
    # if flatfield exists, then print the number of flatfield frames we have.
    if (filter_name in dict_filters_flat):
        print ("    Yes, we do have flatfield frames for %s band filter." \
               % (filter_name) )
        print ("    Number of %s band raw flatfield frames = %d" \
               % (filter_name, dict_filters_flat[filter_name]) )
    # if flatfield does not exist, then print an error message.
    else:
        print ("    No, we do not have flatfield frames for %s band filter." \
               % (filter_name) )
        print ("      ERROR! The data set is not complete! Check the data!")
        # setting completeness parameter
        complete = 0
# printing a summary
print ("Do we have all the necessary flatfield frames?")
if (complete):
    print ("  Yes, looks OK.")
else:
    print ("  No, some data are missing. Check the data.")
```

Execute the script, and check the data.

```
% chmod a+x ao2021_s08_03.py
% ./ao2021_s08_03.py -h
usage: ao2021_s08_03.py [-h] [-f FILTER] [-t TYPE] files [files ...]

Checking filters used for the observation

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -f FILTER, --filter FILTER
                        FITS keyword for filter name
  -t TYPE, --type TYPE  FITS keyword for data type

% ./ao2021_s08_03.py data_ao2021_s08/*.fits
List of filters used for acquiring object frames:
  gp_Astrodon_2019 (19 files)
    Do we have flatfield for gp_Astrodon_2019 band filter?
    Yes, we do have flatfield frames for gp_Astrodon_2019 band filter.
    Number of gp_Astrodon_2019 band raw flatfield frames = 20
  ip_Astrodon_2019 (19 files)
    Do we have flatfield for ip_Astrodon_2019 band filter?
    Yes, we do have flatfield frames for ip_Astrodon_2019 band filter.
    Number of ip_Astrodon_2019 band raw flatfield frames = 20
  rp_Astrodon_2019 (126 files)
    Do we have flatfield for rp_Astrodon_2019 band filter?
    Yes, we do have flatfield frames for rp_Astrodon_2019 band filter.
    Number of rp_Astrodon_2019 band raw flatfield frames = 20
Do we have all the necessary flatfield frames?
  Yes, looks OK.
```

If the data set is not self-consistent, you see a message like below.

```
% ./ao2021_s08_03.py data_ao2021_s08/lot_20210215_0[0-3]*.fits
List of filters used for acquiring object frames:
  gp_Astrodon_2019 (19 files)
    Do we have flatfield for gp_Astrodon_2019 band filter?
    Yes, we do have flatfield frames for gp_Astrodon_2019 band filter.
    Number of gp_Astrodon_2019 band raw flatfield frames = 20
  ip_Astrodon_2019 (19 files)
    Do we have flatfield for ip_Astrodon_2019 band filter?
    No, we do not have flatfield frames for ip_Astrodon_2019 band filter.
      ERROR! The data set is not complete! Check the data!
  rp_Astrodon_2019 (126 files)
    Do we have flatfield for rp_Astrodon_2019 band filter?
    Yes, we do have flatfield frames for rp_Astrodon_2019 band filter.
    Number of rp_Astrodon_2019 band raw flatfield frames = 16
Do we have all the necessary flatfield frames?
  No, some data are missing. Check the data.
```

3.4 Listing exposure time of object and flatfield frames

Make a Python script to scan all the FITS files, and generate a list of exposure time used to acquire object frames and flatfield frames.

Python Code 4: ao2021_s08_04.py

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

# importing argparse module
import argparse

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'Checking exposure time of object and flatfield frames'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
default_exptime_keyword = 'EXPTIME'
default_datatype_keyword = 'IMAGETYP'
parser.add_argument ('-e', '--exptime', default=default_exptime_keyword, \
                     help='FITS keyword for exposure time')
parser.add_argument ('-t', '--type', default=default_datatype_keyword, \
                     help='FITS keyword for data type')
parser.add_argument ('files', nargs='+', help='FITS files')

# command-line argument analysis
```

```
args = parser.parse_args ()

# input parameters
exptime_keyword  = args.exptime
datatype_keyword = args.type
files            = args.files

# declaring a dictionary for filter names
dict_exptime_object = {}
dict_exptime_flat   = {}

# processing FITS files
for fits in files:
    # if the extension of the file is not ’.fits’, the we skip
    if not (fits[-5:] == ’.fits’):
        continue

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

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[datatype_keyword]

    # if the data type is not "LIGHT" or "FLAT", the we skip the file
    if not ( (datatype == ’LIGHT’) or (datatype == ’FLAT’) ):
        continue

    # exposure time
    exptime = header[exptime_keyword]

    if (datatype == ’LIGHT’):
        # object frames
        if not (exptime in dict_exptime_object):
            # appending exptime to the dictionary "dict_exptime_object"
            dict_exptime_object[exptime] = 1
        else:
            # add 1
            dict_exptime_object[exptime] += 1
    elif (datatype == ’FLAT’):
        # flatfield frames
        if not (exptime in dict_exptime_flat):
            # appending exptime to the dictionary "dict_exptime_flat"
            dict_exptime_flat[exptime] = 1
        else:
            # add 1
            dict_exptime_flat[exptime] += 1

# printing a summary
print ("Exposure time summary information:")
print ("  object frames:")
for exptime in sorted (dict_exptime_object.keys () ):
    print ("    %8.3f sec exposure ==> %4d frames" \
           % (float (exptime), dict_exptime_object[exptime]) )
```

```
print ("  flatfield frames:")
for exptime in sorted (dict_exptime_flat.keys () ):
    print ("    %8.3f sec exposure ==> %4d frames" \
           % (float (exptime), dict_exptime_flat[exptime]) )
print ("  List of exposure time used to acquire data:")
list_exptime = list ( dict_exptime_object.keys () ) \
    + list ( dict_exptime_flat.keys () )
for exptime in sorted (list_exptime):
    print ("    %8.3f sec" % float (exptime) )
```

Run the script.

```
% chmod a+x ao2021_s08_04.py
% ./ao2021_s08_04.py -h
usage: ao2021_s08_04.py [-h] [-e EXPTIME] [-t TYPE] files [files ...]

Checking exposure time of object and flatfield frames

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -e EXPTIME, --exptime EXPTIME
                        FITS keyword for exposure time
  -t TYPE, --type TYPE  FITS keyword for data type

% ./ao2021_s08_04.py data_ao2021_s08/*.fits
Exposure time summary information:
  object frames:
      10.000 sec exposure ==>   42 frames
      60.000 sec exposure ==>  116 frames
     180.000 sec exposure ==>    6 frames
  flatfield frames:
       5.000 sec exposure ==>   40 frames
      30.000 sec exposure ==>   20 frames
  List of exposure time used to acquire data:
       5.000 sec
      10.000 sec
      30.000 sec
      60.000 sec
     180.000 sec
```

Now we know that 5, 10, 30, 60, and 180 sec exposure time were used to acquire data.

3.5 Checking whether we have dark frames

If we take 30-sec object or flatfield frames, then 30-sec dark frames must exist. Make a Python script to check whether we have all the necessary dark frames.

Python Code 5: ao2021_s08_05.py

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

# importing argparse module
import argparse

# importing astropy module
```

```
import astropy.io.fits

# construction of parser object
desc = ’Checking exposure time of object and flatfield frames’
parser = argparse.ArgumentParser (description=desc)

# adding arguments
default_exptime_keyword = ’EXPTIME’
default_datatype_keyword = ’IMAGETYP’
parser.add_argument (’-e’, ’--exptime’, default=default_exptime_keyword, \
                     help=’FITS keyword for exposure time’)
parser.add_argument (’-t’, ’--type’, default=default_datatype_keyword, \
                     help=’FITS keyword for data type’)
parser.add_argument (’files’, nargs=’+’, help=’FITS files’)

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

# input parameters
exptime_keyword  = args.exptime
datatype_keyword = args.type
files            = args.files

# declaring a dictionary for filter names
dict_exptime_object = {}
dict_exptime_flat   = {}
dict_exptime_dark   = {}

# processing FITS files
for fits in files:
    # if the extension of the file is not ’.fits’, the we skip
    if not (fits[-5:] == ’.fits’):
        continue

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

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[datatype_keyword]

    # if the data type is not "LIGHT" or "FLAT" or "DARK", the we skip the file
    if not ( (datatype == ’LIGHT’) or (datatype == ’FLAT’) \
             or (datatype == ’DARK’) ):
        continue

    # exposure time
    exptime = header[exptime_keyword]

    if (datatype == ’LIGHT’):
        # object frames
        if not (exptime in dict_exptime_object):
            # appending exptime to the dictionary "dict_exptime_object"
            dict_exptime_object[exptime] = 1
        else:
            # add 1
            dict_exptime_object[exptime] += 1
    elif (datatype == 'FLAT'):
        # flatfield frames
        if not (exptime in dict_exptime_flat):
            # appending exptime to the dictionary "dict_exptime_flat"
            dict_exptime_flat[exptime] = 1
        else:
            # add 1
            dict_exptime_flat[exptime] += 1
    elif (datatype == 'DARK'):
        # dark frames
        if not (exptime in dict_exptime_dark):
            # appending exptime to the dictionary "dict_exptime_dark"
            dict_exptime_dark[exptime] = 1
        else:
            # add 1
            dict_exptime_dark[exptime] += 1

# printing a summary
print ("Exposure time summary information:")
print ("  object frames:")
for exptime in sorted (dict_exptime_object.keys () ):
    print ("    %.3f sec exposure ==> %4d frames" \
           % (float (exptime), dict_exptime_object[exptime]) )
    print ("      Do we have dark frames of %.3f sec exposure?" \
           % float (exptime) )
    if (exptime in dict_exptime_dark):
        print ("      Yes, we do have dark frames of %.3f sec exposure." \
               % float (exptime) )
        print ("      We have %d frames of %.3f sec dark frames." \
               % (dict_exptime_dark[exptime], float (exptime) ) )
    else:
        print ("      No, we do not have dark frames of %.3f sec exposure." \
               % float (exptime) )
        print ("      ERROR! Check the data.")
print ("  flatfield frames:")
for exptime in sorted (dict_exptime_flat.keys () ):
    print ("    %.3f sec exposure ==> %4d frames" \
           % (float (exptime), dict_exptime_flat[exptime]) )
    print ("      Do we have dark frames of %.3f sec exposure?" \
           % float (exptime) )
    if (exptime in dict_exptime_dark):
        print ("      Yes, we do have dark frames of %.3f sec exposure." \
               % float (exptime) )
        print ("      We have %d frames of %.3f sec dark frames." \
               % (dict_exptime_dark[exptime], float (exptime) ) )
    else:
        print ("      No, we do not have dark frames of %.3f sec exposure." \
               % float (exptime) )
        print ("      ERROR! Check the data.")
print ("  Summary of dark frames:")
list_exptime = list ( dict_exptime_object.keys () ) \
    + list ( dict_exptime_flat.keys () )
complete = 1
for exptime in sorted (list_exptime):
    if (exptime in dict_exptime_dark):
        print ("    %8.3f sec dark frames ==> %4d frames are found." \
               % (float (exptime), dict_exptime_dark[exptime] ) )
    else:
        print ("    %8.3f sec dark frames ==> NOT found, check the data!" \
               % (float (exptime) ) )
        complete = 0
if (complete):
    print ("Looks OK. All the necessary dark frames exist.")
else:
    print ("Some dark frames are missing. Check the data!")
```

Run the script.

```
% chmod a+x ao2021_s08_05.py
% ./ao2021_s08_05.py data_ao2021_s08/*.fits
Exposure time summary information:
  object frames:
    10.000 sec exposure ==>   42 frames
      Do we have dark frames of 10.000 sec exposure?
      Yes, we do have dark frames of 10.000 sec exposure.
      We have 10 frames of 10.000 sec dark frames.
    60.000 sec exposure ==>  116 frames
      Do we have dark frames of 60.000 sec exposure?
      Yes, we do have dark frames of 60.000 sec exposure.
      We have 10 frames of 60.000 sec dark frames.
    180.000 sec exposure ==>    6 frames
      Do we have dark frames of 180.000 sec exposure?
      Yes, we do have dark frames of 180.000 sec exposure.
      We have 10 frames of 180.000 sec dark frames.
  flatfield frames:
    5.000 sec exposure ==>   40 frames
      Do we have dark frames of 5.000 sec exposure?
      Yes, we do have dark frames of 5.000 sec exposure.
      We have 50 frames of 5.000 sec dark frames.
    30.000 sec exposure ==>   20 frames
      Do we have dark frames of 30.000 sec exposure?
      Yes, we do have dark frames of 30.000 sec exposure.
      We have 30 frames of 30.000 sec dark frames.
  Summary of dark frames:
       5.000 sec dark frames ==>   50 frames are found.
      10.000 sec dark frames ==>   10 frames are found.
      30.000 sec dark frames ==>   30 frames are found.
      60.000 sec dark frames ==>   10 frames are found.
     180.000 sec dark frames ==>   10 frames are found.
Looks OK. All the necessary dark frames exist.
```

We have all the necessary dark frames, and the data set is self-consistent.
For the case of non-self-consistent data set, you see a message like below.

```
% ./ao2021_s08_05.py data_ao2021_s08/lot_20210215_0[0-5]*
Exposure time summary information:
  object frames:
    10.000 sec exposure ==>   42 frames
      Do we have dark frames of 10.000 sec exposure?
      Yes, we do have dark frames of 10.000 sec exposure.
      We have 10 frames of 10.000 sec dark frames.
    60.000 sec exposure ==>  116 frames
      Do we have dark frames of 60.000 sec exposure?
      No, we do not have dark frames of 60.000 sec exposure.
      ERROR! Check the data.
    180.000 sec exposure ==>    6 frames
      Do we have dark frames of 180.000 sec exposure?
      No, we do not have dark frames of 180.000 sec exposure.
      ERROR! Check the data.
  flatfield frames:
    5.000 sec exposure ==>   40 frames
      Do we have dark frames of 5.000 sec exposure?
      Yes, we do have dark frames of 5.000 sec exposure.
      We have 50 frames of 5.000 sec dark frames.
    30.000 sec exposure ==>   20 frames
      Do we have dark frames of 30.000 sec exposure?
      Yes, we do have dark frames of 30.000 sec exposure.
      We have 28 frames of 30.000 sec dark frames.
  Summary of dark frames:
       5.000 sec dark frames ==>   50 frames are found.
      10.000 sec dark frames ==>   10 frames are found.
      30.000 sec dark frames ==>   28 frames are found.
      60.000 sec dark frames ==> NOT found, check the data!
     180.000 sec dark frames ==> NOT found, check the data!
Some dark frames are missing. Check the data!
```

4 Dark frames

4.1 Examining dark frames

Make a Python script to examine dark frames.

Python Code 6: ao2021_s08_06.py

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

# importing argparse module
import argparse

# importing numpy module
import numpy
import numpy.ma

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

# construction pf parser object
desc = 'calculating statistical information of FITS files'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
list_datatype  = ['BIAS', 'DARK', 'FLAT', 'LIGHT']
list_rejection = ['NONE', 'sigclip']
parser.add_argument ('-e', '--exptime', type=float, default=0.0, \
                     help='exposure time')
parser.add_argument ('-f', '--filter', default='', help='filter name')
parser.add_argument ('-d', '--datatype', default='BIAS', \
                     choices=list_datatype, help='data type')
parser.add_argument ('-r', '--rejection', default='NONE', \
                     choices=list_rejection, help='rejection algorithm')
```

```
parser.add_argument (’-t’, ’--threshold’, type=float, default=4.0, \
                     help=’threshold for sigma clipping’)
parser.add_argument (’-n’, ’--maxiters’, type=int, default=10, \
                     help=’maximum number of iterations’)
parser.add_argument (’files’, nargs=’+’, help=’FITS files’)

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

# input parameters
exptime0   = args.exptime
filter0    = args.filter
datatype0  = args.datatype
rejection  = args.rejection
threshold  = args.threshold
maxiters   = args.maxiters
list_files = args.files

# printing information
print ("# Data search condition:")
print ("#   data type = %s" % datatype0)
print ("#   exptime   = %.3f sec" % exptime0)
print ("#   filter    = \"%s\"" % filter0)
print ("# Input parameters")
print ("#   rejection algorithm = %s" % rejection)
print ("#   threshold of sigma-clipping = %f" % threshold)

# printing header
print ("#")
print ("# %-22s %8s %8s %8s %8s %8s %8s" \
       % (’file name’, ’npix’, ’mean’, ’median’, ’stddev’, ’min’, ’max’) )
print ("#")

# scanning files
for file_fits in list_files:
    # if the file is not a FITS file, then skip
    if not (file_fits[-5:] == ’.fits’):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # FITS keywords
    datatype = header[’IMAGETYP’]
    exptime  = header[’EXPTIME’]
    if ( (datatype == ’LIGHT’) or (datatype == ’FLAT’) ):
        filter = header[’FILTER’]
    else:
        filter = ’NONE’

    # calculate statistical information?
    calc = 0
```

```
        # check of FITS header
        if ( (datatype == 'LIGHT') or (datatype == 'FLAT') ):
            if ( (datatype == datatype0) and (exptime == exptime0) \
                 and (filter == filter0) ):
                # we do calculate statistical information
                calc = 1
        elif ( (datatype == 'BIAS') or (datatype == 'DARK') ):
            if ( (datatype == datatype0) and (exptime == exptime0) ):
                # we do calculate statistical information
                calc = 1

        # skip, if calc == 0
        if (calc == 0):
            continue

        # opening FITS file
        hdu_list = astropy.io.fits.open (file_fits)

        # image of primary HDU
        data0 = hdu_list[0].data

        # closing FITS file
        hdu_list.close ()

        # calculations

        # for no rejection algorithm
        if (rejection == 'NONE'):
            # making a masked array
            data1 = numpy.ma.array (data0, mask=False)
        # for sigma clipping algorithm
        elif (rejection == 'sigclip'):
            data1 = numpy.ma.array (data0, mask=False)
            # iterations
            for j in range (maxiters):
                # number of usable pixels of previous iterations
                npix_prev = len (numpy.ma.compressed (data1) )
                # calculation of median
                median = numpy.ma.median (data1)
                # calculation of standard deviation
                stddev = numpy.ma.std (data1)
                # lower threshold
                low = median - threshold * stddev
                # higher threshold
                high = median + threshold * stddev
                # making a mask
                mask = (data1 < low) | (data1 > high)
                # making a masked array
                data1 = numpy.ma.array (data0, mask=mask)
                # number of usable pixels
                npix_now = len (numpy.ma.compressed (data1) )
                # leaving the loop, if number of usable pixels do not change
                if (npix_now == npix_prev):
                    break

        # calculation of mean, median, stddev, min, and max
        mean   = numpy.ma.mean (data1)
        median = numpy.ma.median (data1)
        stddev = numpy.ma.std (data1)
```

```
    vmin   = numpy.ma.min (data1)
    vmax   = numpy.ma.max (data1)

    # number of pixels
    npix = len (data1.compressed () )

    # file name
    filename = file_fits.split ('/') [-1]

    # printing result
    print ("%-24s %8d %8.2f %8.2f %8.2f %8.2f %8.2f" \
           % (filename, npix, mean, median, stddev, vmin, vmax) )
```

Check 180-sec dark frames.

```
% chmod a+x ao2021_s08_06.py
% ./ao2021_s08_06.py -h
usage: ao2021_s08_06.py [-h] [-e EXPTIME] [-f FILTER]
                        [-d {BIAS,DARK,FLAT,LIGHT}] [-r {NONE,sigclip}]
                        [-t THRESHOLD] [-n MAXITERS]
                        files [files ...]

calculating statistical information of FITS files

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -e EXPTIME, --exptime EXPTIME
                        exposure time
  -f FILTER, --filter FILTER
                        filter name
  -d {BIAS,DARK,FLAT,LIGHT}, --datatype {BIAS,DARK,FLAT,LIGHT}
                        data type
  -r {NONE,sigclip}, --rejection {NONE,sigclip}
                        rejection algorithm
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for sigma clipping
  -n MAXITERS, --maxiters MAXITERS
                        maximum number of iterations

% ./ao2021_s08_06.py -r sigclip -e 180 -d DARK data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 180.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#
# file name                     npix     mean   median   stddev      min      max
#
lot_20210215_0672.fits    4193623   605.99   606.00     7.94   575.00   637.00
lot_20210215_0673.fits    4193551   605.45   605.00     7.95   574.00   636.00
lot_20210215_0674.fits    4193574   605.88   606.00     7.94   575.00   637.00
lot_20210215_0675.fits    4193491   606.07   606.00     7.94   575.00   637.00
lot_20210215_0676.fits    4193752   605.78   606.00     7.95   575.00   637.00
```

```
lot_20210215_0677.fits    4193560   606.23   606.00     7.95   575.00   637.00
lot_20210215_0678.fits    4193495   605.73   606.00     7.94   575.00   637.00
lot_20210215_0679.fits    4193353   606.76   607.00     7.95   576.00   638.00
lot_20210215_0680.fits    4193599   606.29   606.00     7.94   575.00   637.00
lot_20210215_0681.fits    4193533   606.53   606.00     7.95   575.00   637.00
```

All the 10 180-sec dark frames look fine.
Next, check 60-sec dark frames.

```
% ./ao2021_s08_06.py -r sigclip -e 60 -d DARK data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 60.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#
# file name                  npix     mean   median   stddev      min      max
#
lot_20210215_0612.fits    4193823   605.78   606.00     7.97   575.00   637.00
lot_20210215_0613.fits    4193829   606.46   606.00     7.96   575.00   637.00
lot_20210215_0614.fits    4193827   606.73   607.00     7.96   576.00   638.00
lot_20210215_0615.fits    4193898   605.88   606.00     7.94   575.00   637.00
lot_20210215_0616.fits    4193802   606.55   606.00     7.95   575.00   637.00
lot_20210215_0617.fits    4193812   606.20   606.00     7.95   575.00   637.00
lot_20210215_0618.fits    4193819   605.62   606.00     7.96   575.00   637.00
lot_20210215_0619.fits    4193844   605.47   605.00     7.94   574.00   636.00
lot_20210215_0620.fits    4193815   606.30   606.00     7.94   575.00   637.00
lot_20210215_0621.fits    4193743   606.71   607.00     7.95   576.00   638.00
```

All the 10 60-sec dark frames look fine.
Next, check 30-sec dark frames.

```
% ./ao2021_s08_06.py -r sigclip -e 30 -d DARK data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 30.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#
# file name                  npix     mean   median   stddev      min      max
#
lot_20210215_0296.fits    4194014   605.91   606.00     8.06   574.00   638.00
lot_20210215_0299.fits    4193946   606.10   606.00     8.06   574.00   638.00
lot_20210215_0302.fits    4193950   606.07   606.00     8.07   574.00   638.00
lot_20210215_0305.fits    4193876   605.56   606.00     8.07   574.00   638.00
lot_20210215_0308.fits    4193957   605.70   606.00     8.08   574.00   638.00

.....

lot_20210215_0597.fits    4193939   605.86   606.00     7.96   575.00   637.00
lot_20210215_0598.fits    4193903   605.72   606.00     7.96   575.00   637.00
lot_20210215_0599.fits    4193891   606.25   606.00     7.95   575.00   637.00
lot_20210215_0600.fits    4193903   606.12   606.00     7.95   575.00   637.00
lot_20210215_0601.fits    4193955   606.13   606.00     7.96   575.00   637.00
```

Looks good.
Next, check 10-sec dark frames.

```
% ./ao2021_s08_06.py -r sigclip -e 10 -d DARK data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 10.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#
# file name                 npix     mean   median   stddev      min      max
#
lot_20210215_0532.fits    4194038   606.18   606.00     8.00   574.00   638.00
lot_20210215_0533.fits    4193915   606.22   606.00     7.99   575.00   637.00
lot_20210215_0534.fits    4193914   606.01   606.00     8.00   575.00   637.00
lot_20210215_0535.fits    4193920   605.72   606.00     7.99   575.00   637.00
lot_20210215_0536.fits    4193871   605.16   605.00     7.98   574.00   636.00
lot_20210215_0537.fits    4193965   606.02   606.00     7.97   575.00   637.00
lot_20210215_0538.fits    4193915   605.80   606.00     7.98   575.00   637.00
lot_20210215_0539.fits    4193951   605.91   606.00     7.98   575.00   637.00
lot_20210215_0540.fits    4193904   606.45   606.00     7.97   575.00   637.00
lot_20210215_0541.fits    4193956   605.54   606.00     7.97   575.00   637.00
```

Looks OK.
Check 5-sec dark frames.

```
% ./ao2021_s08_06.py -r sigclip -e 5 -d DARK data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 5.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#
# file name                 npix     mean   median   stddev      min      max
#
lot_20210215_0356.fits    4193986   606.08   606.00     8.08   574.00   638.00
lot_20210215_0359.fits    4193933   605.35   605.00     8.08   573.00   637.00
lot_20210215_0362.fits    4193986   605.18   605.00     8.09   573.00   637.00
lot_20210215_0365.fits    4193965   606.06   606.00     8.09   574.00   638.00
lot_20210215_0368.fits    4193962   606.09   606.00     8.09   574.00   638.00

.....

lot_20210215_0517.fits    4194077   606.04   606.00     8.00   574.00   638.00
lot_20210215_0518.fits    4194067   606.54   606.00     8.01   574.00   638.00
lot_20210215_0519.fits    4194056   605.75   606.00     8.00   574.00   638.00
lot_20210215_0520.fits    4194098   605.55   606.00     8.00   574.00   638.00
lot_20210215_0521.fits    4193920   606.58   607.00     7.99   576.00   638.00
```

Looks fine.

4.2 Combining dark frames

Make a Python script to combine dark frames.

Python Code 7: ao2021_s08_07.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing datetime module
import datetime

# importing numpy module
import numpy
import numpy.ma

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

# construction pf parser object
desc = 'combining dark frames'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
list_datatype  = ['BIAS', 'DARK', 'FLAT', 'LIGHT']
list_rejection = ['NONE', 'sigclip']
list_cenfunc   = ['mean', 'median']
parser.add_argument ('-e', '--exptime', type=float, default=0.0, \
                     help='exposure time')
parser.add_argument ('-f', '--filter', default='', help='filter name')
parser.add_argument ('-d', '--datatype', default='BIAS', \
                     choices=list_datatype, help='data type')
parser.add_argument ('-r', '--rejection', default='NONE', \
                     choices=list_rejection, \
                     help='rejection algorithm (default: NONE)')
parser.add_argument ('-t', '--threshold', type=float, default=4.0, \
                     help='threshold for sigma clipping (default: 4.0)')
parser.add_argument ('-n', '--maxiters', type=int, default=10, \
                     help='maximum number of iterations (default: 10)')
parser.add_argument ('-c', '--cenfunc', choices=list_cenfunc, \
                     default='median', \
                     help='method to estimate centre value (default: median)')
parser.add_argument ('-o', '--output', default='', help='output file name')
parser.add_argument ('files', nargs='+', help='FITS files')

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

# input parameters
exptime0    = args.exptime
filter0     = args.filter
datatype0   = args.datatype
rejection   = args.rejection
```

```
threshold   = args.threshold
maxiters    = args.maxiters
cenfunc     = args.cenfunc
file_output = args.output
list_files  = args.files

# examination of output file name
if (file_output == ''):
    print ("Output file name must be given.")
    sys.exit ()
if not (file_output[-5:] == '.fits'):
    print ("Output file must be a FITS file.")
    sys.exit ()

# command name
command = sys.argv[0]

# declaration of list
list_target_files = []

# date/time
now = datetime.datetime.now ().isoformat ()

# printing information
print ("# Data search condition:")
print ("#   data type = %s" % datatype0)
print ("#   exptime   = %.3f sec" % exptime0)
print ("#   filter    = \"%s\"" % filter0)
print ("# Input parameters")
print ("#   rejection algorithm = %s" % rejection)
print ("#   threshold of sigma-clipping = %f" % threshold)
print ("#   maximum number of iterations = %d" % maxiters)

# printing status
print ("#")
print ("# Now scanning data...")

# scanning files
for file_fits in list_files:
    # if the file is not a FITS file, then skip
    if not (file_fits[-5:] == '.fits'):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # FITS keywords
    datatype = header['IMAGETYP']
    exptime  = header['EXPTIME']
    if ( (datatype == 'LIGHT') or (datatype == 'FLAT') ):
        filter = header['FILTER']
    else:
        filter = 'NONE'
    # calculate statistical information?
    calc = 0

    # check of FITS header
    if ( (datatype == 'LIGHT') or (datatype == 'FLAT') ):
        if ( (datatype == datatype0) and (exptime == exptime0) \
             and (filter == filter0) ):
            # appending file name to the list
            list_target_files.append (file_fits)
    elif ( (datatype == 'BIAS') or (datatype == 'DARK') ):
        if ( (datatype == datatype0) and (exptime == exptime0) ):
            # appending file name to the list
            list_target_files.append (file_fits)

# printing status
print ("#")
print ("# Finished scanning files")
print ("#   %d files are found for combining" % len (list_target_files) )

# checking number of target files
if ( len (list_target_files) < 2 ):
    print ("number of target files must be greater than 1.")
    sys.exit ()

print ("#")
print ("# Target files:")
for file_fits in list_target_files:
    print ("#   %s" % file_fits)

# counter
i = 0

# printing status
print ("#")
print ("# Reading image data...")

# reading dark frames
for file_fits in list_target_files:
    # printing status
    print ("#   %04d: \"%s\"" % (i + 1, file_fits) )

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

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

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

    # closing FITS file
    hdu_list.close ()

    # masking a mask

    # for no rejection algorithm
    if (rejection == 'NONE'):
        # making a mask
        mask = numpy.zeros_like (data)
    # for sigma clipping algorithm
    elif (rejection == 'sigclip'):
        mdata = numpy.ma.array (data, mask=False)
        # iterations
        for j in range (maxiters):
            # number of usable pixels of previous iterations
            npix_prev = len ( numpy.ma.compressed (mdata) )
            # calculation of median
            median = numpy.ma.median (mdata)
            # calculation of standard deviation
            stddev = numpy.ma.std (mdata)
            # lower threshold
            low = median - threshold * stddev
            # higher threshold
            high = median + threshold * stddev
            # making a mask
            mask = (mdata < low) | (mdata > high)
            # masked array
            mdata = numpy.ma.array (data, mask=mask)
            # number of rejected pixels
            npix_now = len ( numpy.ma.compressed (mdata) )
            # leaving the loop, if number of usable pixels do not change
            if (npix_now == npix_prev):
                break

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

    # incrementing "i"
    i += 1

# printing status
print ("#")
print ("# Finished reading image data")

# printing status
print ("#")
print ("# Combining image...")

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

# combining dark frames
if (rejection == 'sigclip'):
    # sigma clipping
    clipped_masked_cube = \
        astropy.stats.sigma_clip (masked_cube, sigma=threshold, \
                                  maxiters=maxiters, cenfunc=cenfunc, \
                                  axis=0, masked=True)
    # combining using average
    combined = numpy.ma.average (clipped_masked_cube, axis=0)
elif (rejection == 'NONE'):
    # combining using average
    combined = numpy.ma.average (masked_cube, axis=0)

# printing status
print ("#")
print ("# Finished combining image")

# printing status
print ("#")
print ("# Writing image into a new FITS file...")
print ("#   output file = %s" % file_output)

# mean of combined image
mean_combined = numpy.ma.mean (combined)

# adding comments to the header
header['history'] = "FITS file created by the command \"%s\"" % (command)
header['history'] = "Updated on %s" % (now)
header['comment'] = "multiple FITS files are combined into a single FITS file"
header['comment'] = "List of combined files:"
for file_fits in list_target_files:
    header['comment'] = "  %s" % (file_fits)
header['comment'] = "Options given:"
header['comment'] = "  rejection = %s" % (rejection)
header['comment'] = "  threshold = %f sigma" % (threshold)
header['comment'] = "  maxiters  = %d" % (maxiters)
header['comment'] = "  cenfunc   = %s" % (cenfunc)

# writing a new FITS file
astropy.io.fits.writeto (file_output, \
                         numpy.ma.filled (combined, fill_value=mean_combined), \
                         header=header)

# printing status
print ("#")
print ("# Finished writing image into a new FITS file")
print ("#")
```

Execute the script. First, combine 180-sec dark frames.

```
% chmod a+x ao2021_s08_07.py
% ./ao2021_s08_07.py -h
usage: ao2021_s08_07.py [-h] [-e EXPTIME] [-f FILTER]
                        [-d {BIAS,DARK,FLAT,LIGHT}] [-r {NONE,sigclip}]
                        [-t THRESHOLD] [-n MAXITERS] [-c {mean,median}]
                        [-o OUTPUT]
                        files [files ...]

combining dark frames

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -e EXPTIME, --exptime EXPTIME
                        exposure time
  -f FILTER, --filter FILTER
                        filter name
  -d {BIAS,DARK,FLAT,LIGHT}, --datatype {BIAS,DARK,FLAT,LIGHT}
                        data type
  -r {NONE,sigclip}, --rejection {NONE,sigclip}
                        rejection algorithm (default: NONE)
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for sigma clipping (default: 4.0)
  -n MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 10)
  -c {mean,median}, --cenfunc {mean,median}
                        method to estimate centre value (default: median)
  -o OUTPUT, --output OUTPUT
                        output file name
% ./ao2021_s08_07.py -o dark_0180.fits -e 180 -d DARK -r sigclip \
? data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 180.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# Now scanning data...
#
# Finished scanning files
#   10 files are found for combining
#
# Target files:
#   data_ao2021_s08/lot_20210215_0672.fits
#   data_ao2021_s08/lot_20210215_0673.fits
#   data_ao2021_s08/lot_20210215_0674.fits
#   data_ao2021_s08/lot_20210215_0675.fits
#   data_ao2021_s08/lot_20210215_0676.fits
#   data_ao2021_s08/lot_20210215_0677.fits
#   data_ao2021_s08/lot_20210215_0678.fits
#   data_ao2021_s08/lot_20210215_0679.fits
#   data_ao2021_s08/lot_20210215_0680.fits
#   data_ao2021_s08/lot_20210215_0681.fits
#
# Reading image data...
#   0001: "data_ao2021_s08/lot_20210215_0672.fits"
#   0002: "data_ao2021_s08/lot_20210215_0673.fits"
#   0003: "data_ao2021_s08/lot_20210215_0674.fits"
#   0004: "data_ao2021_s08/lot_20210215_0675.fits"
#   0005: "data_ao2021_s08/lot_20210215_0676.fits"
#   0006: "data_ao2021_s08/lot_20210215_0677.fits"
#   0007: "data_ao2021_s08/lot_20210215_0678.fits"
#   0008: "data_ao2021_s08/lot_20210215_0679.fits"
#   0009: "data_ao2021_s08/lot_20210215_0680.fits"
#   0010: "data_ao2021_s08/lot_20210215_0681.fits"
#
# Finished reading image data
#
```

```
# Combining image...
#
# Finished combining image
#
# Writing image into a new FITS file...
#   output file = dark_0180.fits
#
# Finished writing image into a new FITS file
#
% ls -l dark_0180.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  4 19:31 dark_0180.fits
```

Next, combine 60-sec dark frames.

```
% ./ao2021_s08_07.py -o dark_0060.fits -e 60 -d DARK -r sigclip \
? data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 60.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# Now scanning data...
#
# Finished scanning files
#   10 files are found for combining
#
# Target files:
#   data_ao2021_s08/lot_20210215_0612.fits
#   data_ao2021_s08/lot_20210215_0613.fits
#   data_ao2021_s08/lot_20210215_0614.fits
#   data_ao2021_s08/lot_20210215_0615.fits
#   data_ao2021_s08/lot_20210215_0616.fits
#   data_ao2021_s08/lot_20210215_0617.fits
#   data_ao2021_s08/lot_20210215_0618.fits
#   data_ao2021_s08/lot_20210215_0619.fits
#   data_ao2021_s08/lot_20210215_0620.fits
#   data_ao2021_s08/lot_20210215_0621.fits
#
# Reading image data...
#   0001: "data_ao2021_s08/lot_20210215_0612.fits"
#   0002: "data_ao2021_s08/lot_20210215_0613.fits"
#   0003: "data_ao2021_s08/lot_20210215_0614.fits"
#   0004: "data_ao2021_s08/lot_20210215_0615.fits"
#   0005: "data_ao2021_s08/lot_20210215_0616.fits"
#   0006: "data_ao2021_s08/lot_20210215_0617.fits"
#   0007: "data_ao2021_s08/lot_20210215_0618.fits"
#   0008: "data_ao2021_s08/lot_20210215_0619.fits"
#   0009: "data_ao2021_s08/lot_20210215_0620.fits"
#   0010: "data_ao2021_s08/lot_20210215_0621.fits"
#
# Finished reading image data
#
# Combining image...
#
```

```
# Finished combining image
#
# Writing image into a new FITS file...
#   output file = dark_0060.fits
#
# Finished writing image into a new FITS file
#
% ls -l dark_*.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  4 19:34 dark_0060.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  4 19:31 dark_0180.fits
```

Combine 30, 10, and 5 sec dark frames. If the memory of your computer is not enough, the Python script may fail to combine data. If the memory is not enough, reduce the number of files to combine.

```
% ./ao2021_s08_07.py -o dark_0030.fits -e 30 -d DARK -r sigclip \
? data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 30.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# Now scanning data...
#
# Finished scanning files
#   30 files are found for combining
#
# Target files:
#   data_ao2021_s08/lot_20210215_0296.fits
#   data_ao2021_s08/lot_20210215_0299.fits
#   data_ao2021_s08/lot_20210215_0302.fits
#   data_ao2021_s08/lot_20210215_0305.fits
#   data_ao2021_s08/lot_20210215_0308.fits

.....

#   data_ao2021_s08/lot_20210215_0597.fits
#   data_ao2021_s08/lot_20210215_0598.fits
#   data_ao2021_s08/lot_20210215_0599.fits
#   data_ao2021_s08/lot_20210215_0600.fits
#   data_ao2021_s08/lot_20210215_0601.fits
#
# Reading image data...
#   0001: "data_ao2021_s08/lot_20210215_0296.fits"
#   0002: "data_ao2021_s08/lot_20210215_0299.fits"
#   0003: "data_ao2021_s08/lot_20210215_0302.fits"
#   0004: "data_ao2021_s08/lot_20210215_0305.fits"
#   0005: "data_ao2021_s08/lot_20210215_0308.fits"

.....

#   0026: "data_ao2021_s08/lot_20210215_0597.fits"
#   0027: "data_ao2021_s08/lot_20210215_0598.fits"
#   0028: "data_ao2021_s08/lot_20210215_0599.fits"
#   0029: "data_ao2021_s08/lot_20210215_0600.fits"
#   0030: "data_ao2021_s08/lot_20210215_0601.fits"
#
# Finished reading image data
#
# Combining image...
#
# Finished combining image
#
# Writing image into a new FITS file...
#   output file = dark_0030.fits
#
# Finished writing image into a new FITS file
#
% ./ao2021_s08_07.py -o dark_0010.fits -e 10 -d DARK -r sigclip \
? data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 10.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# Now scanning data...
#
# Finished scanning files
#   10 files are found for combining
#
# Target files:
#   data_ao2021_s08/lot_20210215_0532.fits
#   data_ao2021_s08/lot_20210215_0533.fits
#   data_ao2021_s08/lot_20210215_0534.fits
#   data_ao2021_s08/lot_20210215_0535.fits
#   data_ao2021_s08/lot_20210215_0536.fits
#   data_ao2021_s08/lot_20210215_0537.fits
#   data_ao2021_s08/lot_20210215_0538.fits
#   data_ao2021_s08/lot_20210215_0539.fits
#   data_ao2021_s08/lot_20210215_0540.fits
#   data_ao2021_s08/lot_20210215_0541.fits
#
# Reading image data...
#   0001: "data_ao2021_s08/lot_20210215_0532.fits"
#   0002: "data_ao2021_s08/lot_20210215_0533.fits"
#   0003: "data_ao2021_s08/lot_20210215_0534.fits"
#   0004: "data_ao2021_s08/lot_20210215_0535.fits"
#   0005: "data_ao2021_s08/lot_20210215_0536.fits"
#   0006: "data_ao2021_s08/lot_20210215_0537.fits"
#   0007: "data_ao2021_s08/lot_20210215_0538.fits"
#   0008: "data_ao2021_s08/lot_20210215_0539.fits"
#   0009: "data_ao2021_s08/lot_20210215_0540.fits"
#   0010: "data_ao2021_s08/lot_20210215_0541.fits"
#
# Finished reading image data
#
# Combining image...
#
# Finished combining image
#
```

```
# Writing image into a new FITS file...
#   output file = dark_0010.fits
#
# Finished writing image into a new FITS file
#
% ./ao2021_s08_07.py -o dark_0005.fits -e 5 -d DARK -r sigclip \
? data_ao2021_s08/*.fits
# Data search condition:
#   data type = DARK
#   exptime   = 5.000 sec
#   filter    = ""
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# Now scanning data...
#
# Finished scanning files
#   50 files are found for combining
#
# Target files:
#   data_ao2021_s08/lot_20210215_0356.fits
#   data_ao2021_s08/lot_20210215_0359.fits
#   data_ao2021_s08/lot_20210215_0362.fits
#   data_ao2021_s08/lot_20210215_0365.fits
#   data_ao2021_s08/lot_20210215_0368.fits

.....

#   data_ao2021_s08/lot_20210215_0517.fits
#   data_ao2021_s08/lot_20210215_0518.fits
#   data_ao2021_s08/lot_20210215_0519.fits
#   data_ao2021_s08/lot_20210215_0520.fits
#   data_ao2021_s08/lot_20210215_0521.fits
#
# Reading image data...
#   0001: "data_ao2021_s08/lot_20210215_0356.fits"
#   0002: "data_ao2021_s08/lot_20210215_0359.fits"
#   0003: "data_ao2021_s08/lot_20210215_0362.fits"
#   0004: "data_ao2021_s08/lot_20210215_0365.fits"
#   0005: "data_ao2021_s08/lot_20210215_0368.fits"

.....

#   0046: "data_ao2021_s08/lot_20210215_0517.fits"
#   0047: "data_ao2021_s08/lot_20210215_0518.fits"
#   0048: "data_ao2021_s08/lot_20210215_0519.fits"
#   0049: "data_ao2021_s08/lot_20210215_0520.fits"
#   0050: "data_ao2021_s08/lot_20210215_0521.fits"
#
# Finished reading image data
#
# Combining image...
#
# Finished combining image
#
# Writing image into a new FITS file...
#   output file = dark_0005.fits
```

```
#
# Finished writing image into a new FITS file
#
% ls -l dark_*.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Apr  4 19:44 dark_0005.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  4 19:41 dark_0010.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  4 19:37 dark_0030.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  4 19:34 dark_0060.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  4 19:31 dark_0180.fits
```

4.3 Checking combined dark frames

Check mean and standard deviation of combined dark frames.

Python Code 8: ao2021_s08_08.py

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

# importing argparse module
import argparse

# importing numpy module
import numpy
import numpy.ma

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

# construction pf parser object
desc = 'calculating statistical information of FITS files'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
list_rejection = ['NONE', 'sigclip']
parser.add_argument ('-r', '--rejection', default='NONE', \
                     choices=list_rejection, help='rejection algorithm')
parser.add_argument ('-t', '--threshold', type=float, default=4.0, \
                     help='threshold for sigma clipping')
parser.add_argument ('-n', '--maxiters', type=int, default=10, \
                     help='maximum number of iterations')
parser.add_argument ('files', nargs='+', help='FITS files')

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

# input parameters
rejection  = args.rejection
threshold  = args.threshold
maxiters   = args.maxiters
list_files = args.files

# printing header
print ("#")
print ("# %-22s %8s %8s %8s %8s %8s %8s" \
       % ('file name', 'npix', 'mean', 'median', 'stddev', 'min', 'max') )
print ("#")

# reading files
```

```
for file_fits in list_files:
    # if the file is not a FITS file, then skip
    if not (file_fits[-5:] == '.fits'):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # image of primary HDU
    data0 = hdu_list[0].data

    # closing FITS file
    hdu_list.close ()

    # calculations

    # for no rejection algorithm
    if (rejection == 'NONE'):
        # making a masked array
        data1 = numpy.ma.array (data0, mask=False)
    # for sigma clipping algorithm
    elif (rejection == 'sigclip'):
        data1 = numpy.ma.array (data0, mask=False)
        # iterations
        for j in range (maxiters):
            # number of usable pixels of previous iterations
            npix_prev = len (numpy.ma.compressed (data1) )
            # calculation of median
            median = numpy.ma.median (data1)
            # calculation of standard deviation
            stddev = numpy.ma.std (data1)
            # lower threshold
            low = median - threshold * stddev
            # higher threshold
            high = median + threshold * stddev
            # making a mask
            mask = (data1 < low) | (data1 > high)
            # making a masked array
            data1 = numpy.ma.array (data0, mask=mask)
            # number of usable pixels
            npix_now = len (numpy.ma.compressed (data1) )
            # leaving the loop, if number of usable pixels do not change
            if (npix_now == npix_prev):
                break

    # calculation of mean, median, stddev, min, and max
    mean   = numpy.ma.mean (data1)
    median = numpy.ma.median (data1)
    stddev = numpy.ma.std (data1)
    vmin   = numpy.ma.min (data1)
    vmax   = numpy.ma.max (data1)

    # number of pixels
    npix = len (data1.compressed () )

    # file name
    filename = file_fits.split ('/') [-1]

    # printing result
    print ("%-24s %8d %8.2f %8.2f %8.2f %8.2f %8.2f" \
           % (filename, npix, mean, median, stddev, vmin, vmax) )
```

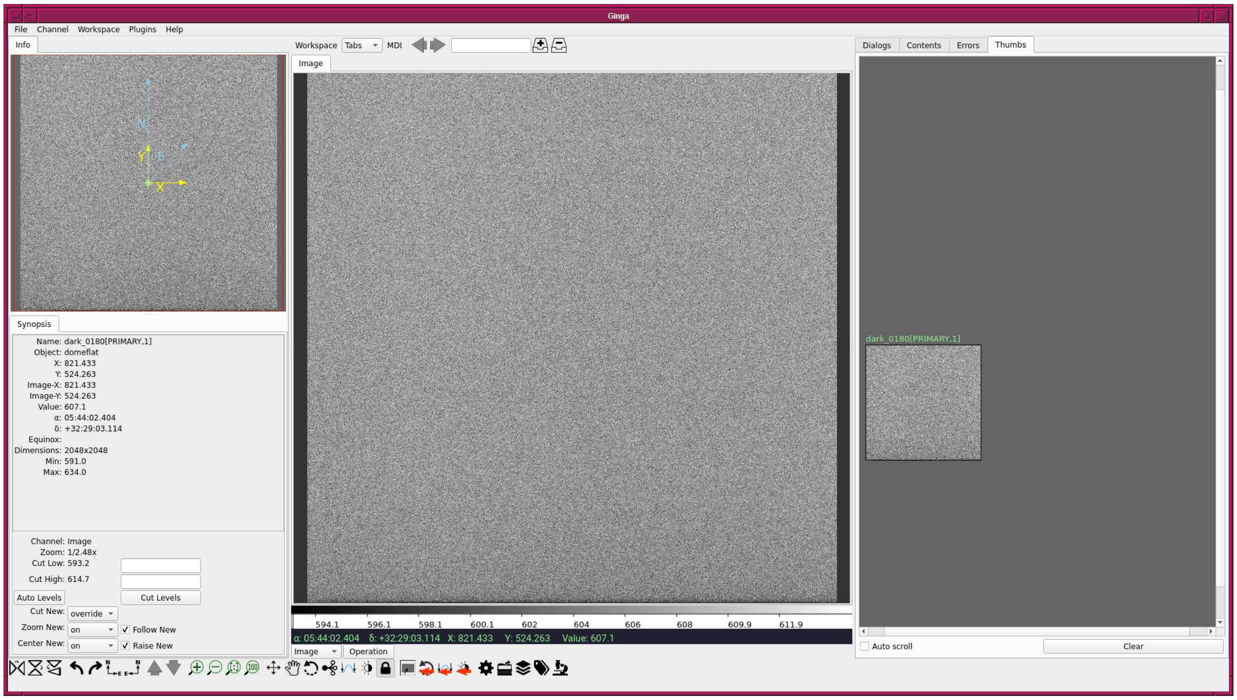
Run the script, and show statistical information of pixel values of combined dark frames.

```
% chmod a+x ao2021_s08_08.py
% ./ao2021_s08_08.py dark_*.fits
#
# file name                     npix     mean   median   stddev      min      max
#
dark_0005.fits               4194304   606.18   606.22     1.28   595.18   630.41
dark_0010.fits               4194304   605.90   605.90     2.58   588.40   620.89
dark_0030.fits               4194304   606.04   606.07     1.59   594.23   629.35
dark_0060.fits               4194304   606.17   606.20     2.58   589.40   633.22
dark_0180.fits               4194304   606.07   606.10     2.58   591.00   634.00
```

4.4 Visual inspection of combined dark frames

Use the FITS image viewer "Ginga" to display combined dark frames on your computer display. Here is an example of showing combined 180-sec dark frame. (Fig. 2)

```
% ginga dark_0180.fits
```



Name: dark_0180[PRIMARY,1]
Object: domeflat
X: 821.433
Y: 524.263
Image-X: 821.433
Image-Y: 524.263
Value: 607.1
α: 05:44:02.404
δ: +32:29:03.114
Equinox:
Dimensions: 2048x2048
Min: 591.0
Max: 634.0

Channel: Image
Zoom: 1/2.48x
Cut Low: 593.2
Cut High: 614.7
Cut New: override
Zoom New: on ☑ Follow New
Center New: on ☑ Raise New

α: 05:44:02.404 δ: +32:29:03.114 X: 821.433 Y: 524.263 Value: 607.1

Image

Figure 2: Combined 180-sec dark frame.

Also, check combined 60-sec, 30-sec, 10-sec, and 5-sec dark frames.

5 Dark subtraction

Carry out dark subtraction.

5.1 Listing target FITS files for dark subtraction

Dark subtraction is needed for object frames and flatfield frames. Search for object and flatfield frames, and make a list of target FITS files for dark subtraction. Here is an example Python script.

Python Code 9: ao2021_s08_09.py

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

# importing argparse module
import argparse

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'listing object and flatfield frames'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
default_filter_keyword   = 'FILTER'
default_datatype_keyword = 'IMAGETYP'
default_exptime_keyword  = 'EXPTIME'
parser.add_argument ('-f', '--filter', default=default_filter_keyword, \
                     help='FITS keyword for filter name')
parser.add_argument ('-d', '--datatype', default=default_datatype_keyword, \
                     help='FITS keyword for data type')
parser.add_argument ('-e', '--exptime', default=default_exptime_keyword, \
                     help='FITS keyword for exposure time')
parser.add_argument ('files', nargs='+', help='FITS files')

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

# input parameters
filter_keyword   = args.filter
datatype_keyword = args.datatype
exptime_keyword  = args.exptime
files            = args.files

# declaring an empty dictionary for storing FITS file information
dict_target = {}

# processing FITS files
for file_fits in files:
    # if the extension of the file is not '.fits', the we skip
    if not (file_fits[-5:] == '.fits'):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[datatype_keyword]
    # exptime
    exptime = header[exptime_keyword]

    # if the data type is not "LIGHT" or "FLAT", the we skip the file
    if not ( (datatype == 'LIGHT') or (datatype == 'FLAT') ):
        continue

    # filter name
    filter_name = header[filter_keyword]

    # appending file name to the dictionary
    dict_target[file_fits] = {}
    dict_target[file_fits]['filter'] = filter_name
    dict_target[file_fits]['exptime'] = exptime

# printing FITS file list
print ("List of FITS files for dark subtraction:")
for file_fits in sorted (dict_target.keys () ):
    print ("  %s (%s, %d sec)" % (file_fits, \
                                  dict_target[file_fits]['filter'],
                                  dict_target[file_fits]['exptime']) )
print ("Total number of FITS files for dark subtraction:")
print ("  %d files" % len (dict_target) )
```

Execute the script, and produce a list of FITS files for dark subtraction.

```
% chmod a+x ao2021_s08_09.py
% ./ao2021_s08_09.py data_ao2021_s08/*.fits
List of FITS files for dark subtraction:
  data_ao2021_s08/lot_20210215_0070.fits (rp_Astrodon_2019, 60 sec)
  data_ao2021_s08/lot_20210215_0071.fits (rp_Astrodon_2019, 60 sec)
  data_ao2021_s08/lot_20210215_0075.fits (rp_Astrodon_2019, 60 sec)
  data_ao2021_s08/lot_20210215_0076.fits (rp_Astrodon_2019, 60 sec)
  data_ao2021_s08/lot_20210215_0077.fits (rp_Astrodon_2019, 60 sec)

.....

  data_ao2021_s08/lot_20210215_0459.fits (ip_Astrodon_2019, 5 sec)
  data_ao2021_s08/lot_20210215_0462.fits (ip_Astrodon_2019, 5 sec)
  data_ao2021_s08/lot_20210215_0465.fits (ip_Astrodon_2019, 5 sec)
  data_ao2021_s08/lot_20210215_0468.fits (ip_Astrodon_2019, 5 sec)
  data_ao2021_s08/lot_20210215_0471.fits (ip_Astrodon_2019, 5 sec)
Total number of FITS files for dark subtraction:
  224 files
```

5.2 Carrying out dark subtraction

Now, we know the target files for dark subtraction. Make a Python script to carry out dark subtraction.

Python Code 10: ao2021_s08_10.py

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

# importing argparse module
import argparse

# importing pathlib module
import pathlib
```

```
# importing sys module
import sys

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'carrying out dark subtraction'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
default_filter_keyword   = 'FILTER'
default_datatype_keyword = 'IMAGETYP'
default_exptime_keyword  = 'EXPTIME'
parser.add_argument ('-f', '--filter', default=default_filter_keyword, \
                     help='FITS keyword for filter name')
parser.add_argument ('-d', '--datatype', default=default_datatype_keyword, \
                     help='FITS keyword for data type')
parser.add_argument ('-e', '--exptime', default=default_exptime_keyword, \
                     help='FITS keyword for exposure time')
parser.add_argument ('files', nargs='+', help='FITS files')

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

# input parameters
filter_keyword   = args.filter
datatype_keyword = args.datatype
exptime_keyword  = args.exptime
files            = args.files

# command name
command = sys.argv[0]

# date/time
now = datetime.datetime.now ().isoformat ()

# declaring an empty dictionary for storing FITS file information
dict_target = {}

# processing FITS files
for file_raw in files:
    # if the extension of the file is not '.fits', the we skip
    if not (file_raw[-5:] == '.fits'):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_raw)

    # header of primary HDU
    header = hdu_list[0].header
    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[datatype_keyword]
    # exptime
    exptime = header[exptime_keyword]

    # if the data type is not "LIGHT" or "FLAT", the we skip the file
    if not ( (datatype == 'LIGHT') or (datatype == 'FLAT') ):
        continue

    # filter name
    filter_name = header[filter_keyword]

    # appending file name to the dictionary
    dict_target[file_raw] = {}
    dict_target[file_raw]['filter'] = filter_name
    dict_target[file_raw]['exptime'] = exptime

# printing FITS file list
print ("# List of FITS files for dark subtraction:")
for file_raw in sorted (dict_target.keys () ):
    print ("#   %s (%s, %d sec)" % (file_raw, \
                                  dict_target[file_raw]['filter'],
                                  dict_target[file_raw]['exptime']) )
print ("# Total number of FITS files for dark subtraction:")
print ("#   %d files" % len (dict_target) )

# dark subtraction

print ("#")
print ("# Processing each FITS file...")
print ("#")

# processing each FITS file
for file_raw in sorted (dict_target.keys () ):
    # file names
    file_subtracted = file_raw.split ('/') [-1] [:-5] + '_d.fits'

    print ("# subtracting dark from %s..." % file_raw)
    print ("#   %s ==> %s" % (file_raw, file_subtracted) )


    # opening FITS file (raw data)
    hdu_list = astropy.io.fits.open (file_raw)

    # header of primary HDU
    header = hdu_list[0].header

    # printing status
    print ("#     reading raw data from \"%s\"..." % file_raw)

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

    # closing FITS file
    hdu_list.close ()
    # exptime
    exptime = header[exptime_keyword]

    # dark file name
    file_dark = "dark_%04d.fits" % (int (exptime) )

    # checking whether dark file exists
    # if dark file does not exist, then stop the script
    path_dark = pathlib.Path (file_dark)
    if not (path_dark.exists () ):
        print ("The dark file \"%s\" is NOT found." % file_dark)
        print ("Check the data!")
        sys.exit ()

    # opening FITS file (dark)
    hdu_list = astropy.io.fits.open (file_dark)

    # header of primary HDU
    header_dark = hdu_list[0].header

    # checking exptime of dark frame
    exptime_dark = header_dark[exptime_keyword]

    # if exptime_dark is not the same as exptime, then stop the script
    if not (exptime == exptime_dark):
        print ("The exposure time of raw frame and dark frame are NOT same.")
        print ("Check the data!")
        sys.exit ()

    # printing status
    print ("#     reading dark data from \"%s\"..." % file_dark)

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

    # closing FITS file
    hdu_list.close ()

    # printing status
    print ("#     subtracting dark from \"%s\"..." % file_raw)

    # dark subtraction
    data_subtracted = data_raw - data_dark

    # adding comments to new FITS file
    header['history'] = "FITS file created by the command \"%s\"" % (command)
    header['history'] = "Updated on %s" % (now)
    header['comment'] = "dark subtraction was carried out"
    header['comment'] = "raw data: %s" % (file_raw)
    header['comment'] = "dark data: %s" % (file_dark)
    header['comment'] = "dark subtracted data: %s" % (file_subtracted)

    # printing status
    print ("#     writing new file \"%s\"..." % file_subtracted)

    # writing a new FITS file
    astropy.io.fits.writeto (file_subtracted, data_subtracted, header=header)
```

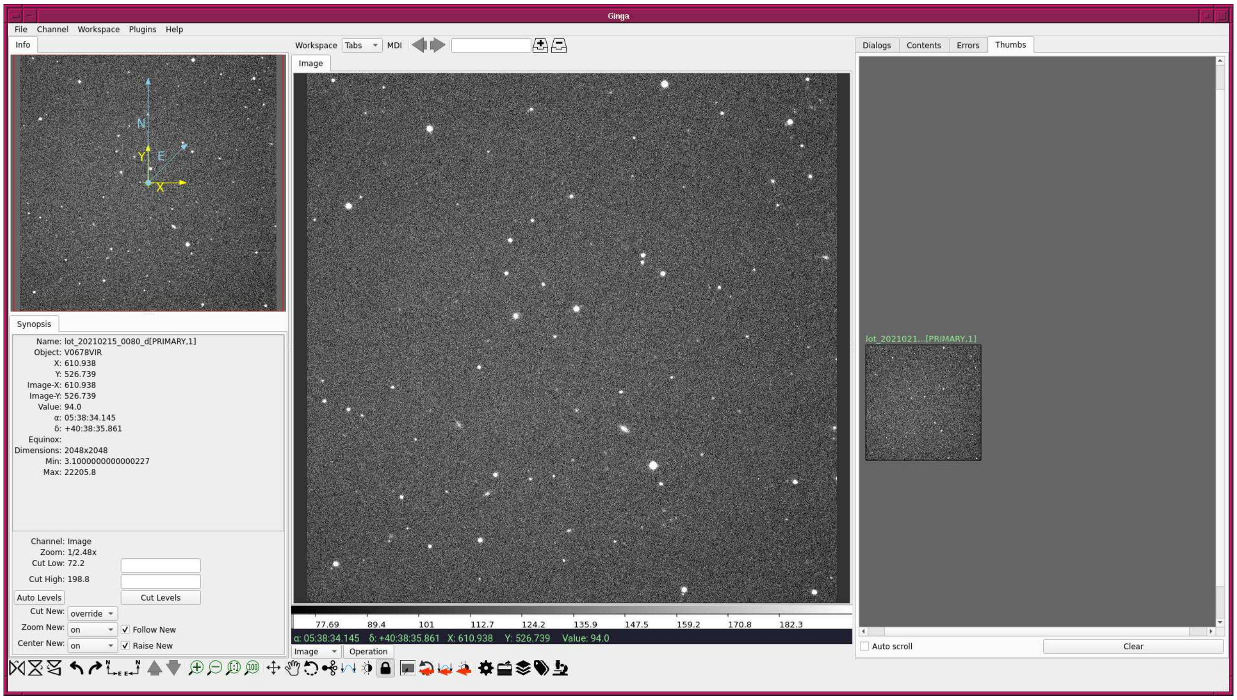
Run the script, and carry out dark subtraction.

```
% chmod a+x ao2021_s08_10.py
% ./ao2021_s08_10.py -h
usage: ao2021_s08_10.py [-h] [-f FILTER] [-d DATATYPE] [-e EXPTIME]
                        files [files ...]

carrying out dark subtraction

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -f FILTER, --filter FILTER
                        FITS keyword for filter name
  -d DATATYPE, --datatype DATATYPE
                        FITS keyword for data type
  -e EXPTIME, --exptime EXPTIME
                        FITS keyword for exposure time

% ./ao2021_s08_10.py data_ao2021_s08/*.fits
# List of FITS files for dark subtraction:
#   data_ao2021_s08/lot_20210215_0070.fits (rp_Astrodon_2019, 60 sec)
#   data_ao2021_s08/lot_20210215_0071.fits (rp_Astrodon_2019, 60 sec)
#   data_ao2021_s08/lot_20210215_0075.fits (rp_Astrodon_2019, 60 sec)
#   data_ao2021_s08/lot_20210215_0076.fits (rp_Astrodon_2019, 60 sec)
#   data_ao2021_s08/lot_20210215_0077.fits (rp_Astrodon_2019, 60 sec)

.....

#   data_ao2021_s08/lot_20210215_0459.fits (ip_Astrodon_2019, 5 sec)
#   data_ao2021_s08/lot_20210215_0462.fits (ip_Astrodon_2019, 5 sec)
#   data_ao2021_s08/lot_20210215_0465.fits (ip_Astrodon_2019, 5 sec)
#   data_ao2021_s08/lot_20210215_0468.fits (ip_Astrodon_2019, 5 sec)
#   data_ao2021_s08/lot_20210215_0471.fits (ip_Astrodon_2019, 5 sec)
# Total number of FITS files for dark subtraction:
#   224 files
#
# Processing each FITS file...
#
# subtracting dark from data_ao2021_s08/lot_20210215_0070.fits...
#   data_ao2021_s08/lot_20210215_0070.fits ==> lot_20210215_0070_d.fits
#     reading raw data from "data_ao2021_s08/lot_20210215_0070.fits"...
#     reading dark data from "dark_0060.fits"...
#     subtracting dark from "data_ao2021_s08/lot_20210215_0070.fits"...
#     writing new file "lot_20210215_0070_d.fits"...
# subtracting dark from data_ao2021_s08/lot_20210215_0071.fits...
#   data_ao2021_s08/lot_20210215_0071.fits ==> lot_20210215_0071_d.fits
#     reading raw data from "data_ao2021_s08/lot_20210215_0071.fits"...
#     reading dark data from "dark_0060.fits"...
#     subtracting dark from "data_ao2021_s08/lot_20210215_0071.fits"...
#     writing new file "lot_20210215_0071_d.fits"...

.....

# subtracting dark from data_ao2021_s08/lot_20210215_0468.fits...
#   data_ao2021_s08/lot_20210215_0468.fits ==> lot_20210215_0468_d.fits
#     reading raw data from "data_ao2021_s08/lot_20210215_0468.fits"...
#     reading dark data from "dark_0005.fits"...
#     subtracting dark from "data_ao2021_s08/lot_20210215_0468.fits"...
#     writing new file "lot_20210215_0468_d.fits"...
# subtracting dark from data_ao2021_s08/lot_20210215_0471.fits...
#   data_ao2021_s08/lot_20210215_0471.fits ==> lot_20210215_0471_d.fits
#     reading raw data from "data_ao2021_s08/lot_20210215_0471.fits"...
#     reading dark data from "dark_0005.fits"...
#     subtracting dark from "data_ao2021_s08/lot_20210215_0471.fits"...
#     writing new file "lot_20210215_0471_d.fits"...
```

5.3 Visual inspection of dark subtracted frames

Use Ginga to carry out visual inspection of dark subtracted frames. Show a dark subtracted object frame using Ginga. An example of dark subtracted object frame is shown in Fig. 3.

```
% ginga lot_20210215_0080_d.fits
```



```
Name: lot_20210215_0080_d[PRIMARY,1]
Object: V0678VIR
X: 610.938
Y: 526.739
Image-X: 610.938
Image-Y: 526.739
Value: 94.0
α: 05:38:34.145
δ: +40:38:35.861
Equinox:
Dimensions: 2048x2048
Min: 3.1000000000000227
Max: 22205.8

Channel: Image
Zoom: 1/2.48x
Cut Low: 72.2
Cut High: 198.8
Cut New: override
Zoom New: on    ☑ Follow New
Center New: on  ☑ Raise New

α: 05:38:34.145   δ: +40:38:35.861   X: 610.938   Y: 526.739   Value: 94.0
```

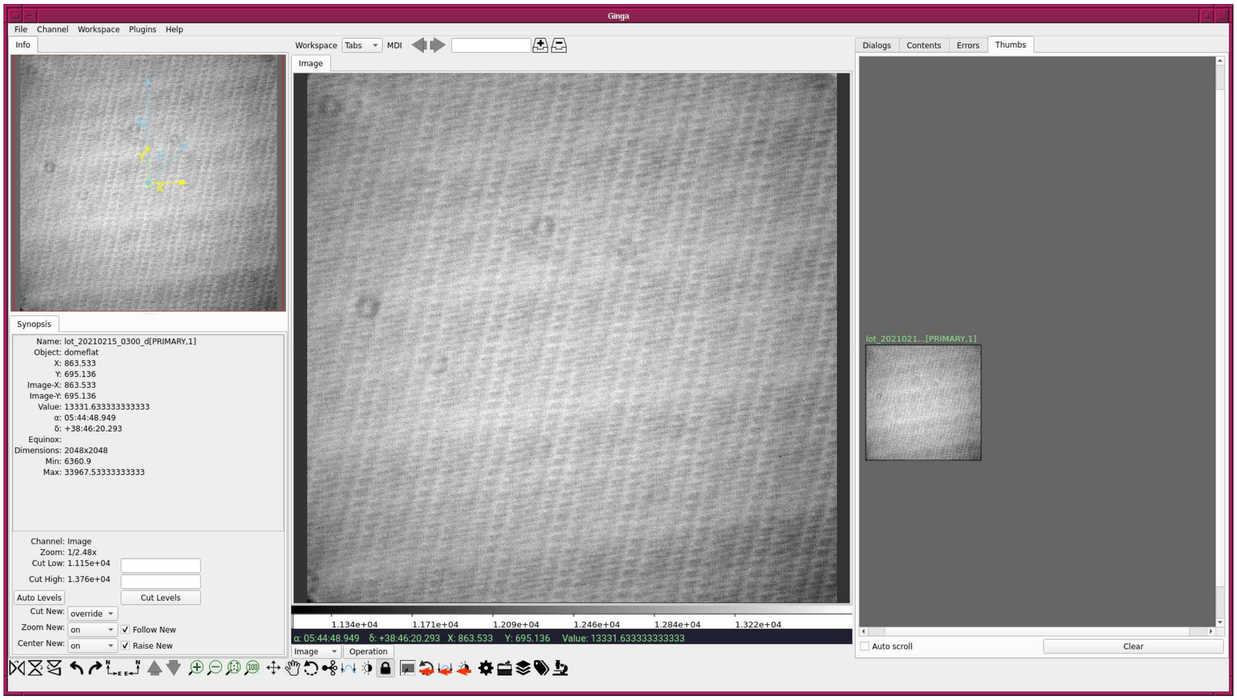
Figure 3: An example of dark subtracted object frame.

Show a dark subtracted flatfield frame using Ginga. An example of dark subtracted flatfield frame is shown in Fig. 4.

```
% ginga lot_20210215_0300_d.fits
```

6 Combining dark-subtracted flatfield

We combine dark-subtracted flatfield frames.



Name: lot_20210215_0300_d[PRIMARY,1]
Object: domeflat
X: 863.533
Y: 695.136
Image-X: 863.533
Image-Y: 695.136
Value: 13331.633333333333
α: 05:44:48.949
δ: +38:46:20.293
Equinox:
Dimensions: 2048x2048
Min: 6360.9
Max: 33967.53333333333

Channel: Image
Zoom: 1/2.48x
Cut Low: 1.115e+04
Cut High: 1.376e+04
Cut New: override
Zoom New: on ☑ Follow New
Center New: on ☑ Raise New

α: 05:44:48.949 δ: +38:46:20.293 X: 863.533 Y: 695.136 Value: 13331.633333333333

Figure 4: An example of dark subtracted flatfield frame.

6.1 Listing dark-subtracted flatfield frames

Make a Python script to list dark subtracted flatfield frames.

Python Code 11: ao2021_s08_11.py

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

# importing argparse module
import argparse

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'listing flatfield frames'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
default_filter_keyword   = 'FILTER'
default_datatype_keyword = 'IMAGETYP'
default_exptime_keyword  = 'EXPTIME'
parser.add_argument ('-f', '--filter', default=default_filter_keyword, \
                     help='FITS keyword for filter name')
parser.add_argument ('-d', '--datatype', default=default_datatype_keyword, \
                     help='FITS keyword for data type')
parser.add_argument ('-e', '--exptime', default=default_exptime_keyword, \
                     help='FITS keyword for exposure time')
parser.add_argument ('files', nargs='+', help='FITS files')

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

# input parameters
```

```
filter_keyword   = args.filter
datatype_keyword = args.datatype
exptime_keyword  = args.exptime
files            = args.files

# declaring an empty dictionary for storing FITS file information
dict_target = {}

# processing FITS files
for file_fits in files:
    # if the extension of the file is not ’.fits’, the we skip
    if not (file_fits[-5:] == ’.fits’):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[datatype_keyword]
    # exptime
    exptime = header[exptime_keyword]
    # date-obs
    date_obs = header[’DATE-OBS’]
    # time-obs
    time_obs = header[’TIME-OBS’]

    # if the data type is not "FLAT", the we skip the file
    if not (datatype == ’FLAT’):
        continue

    # filter name
    filter_name = header[filter_keyword]

    # appending FITS header information to the dictionary
    if not (filter_name in dict_target):
        dict_target[filter_name] = {}
    dict_target[filter_name][file_fits] = {}
    dict_target[filter_name][file_fits][’exptime’]  = exptime
    dict_target[filter_name][file_fits][’date-obs’] = date_obs
    dict_target[filter_name][file_fits][’time-obs’] = time_obs

# printing FITS file list
print ("List of FITS files for constructing combined flatfield:")
for filter_name in sorted (dict_target.keys () ):
    print ("  %s band flatfield:"% filter_name)
    for file_fits in sorted (dict_target[filter_name].keys () ):
        print ("    %s (%d sec data taken at %s on %s)" \
               % (file_fits, dict_target[filter_name][file_fits][’exptime’], \
                  dict_target[filter_name][file_fits][’time-obs’], \
                  dict_target[filter_name][file_fits][’date-obs’]) )
```

Run the script.

```
% chmod a+x ao2021_s08_11.py
% ./ao2021_s08_11.py -h
usage: ao2021_s08_11.py [-h] [-f FILTER] [-d DATATYPE] [-e EXPTIME]
                        files [files ...]

listing flatfield frames

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -f FILTER, --filter FILTER
                        FITS keyword for filter name
  -d DATATYPE, --datatype DATATYPE
                        FITS keyword for data type
  -e EXPTIME, --exptime EXPTIME
                        FITS keyword for exposure time

% ./ao2021_s08_11.py lot_20210215_*_d.fits
List of FITS files for constructing combined flatfield:
  gp_Astrodon_2019 band flatfield:
    lot_20210215_0294_d.fits (30 sec data taken at 20:29:19 on 2021-02-15)
    lot_20210215_0297_d.fits (30 sec data taken at 20:30:43 on 2021-02-15)
    lot_20210215_0300_d.fits (30 sec data taken at 20:32:02 on 2021-02-15)
    lot_20210215_0303_d.fits (30 sec data taken at 20:33:19 on 2021-02-15)
    lot_20210215_0306_d.fits (30 sec data taken at 20:34:39 on 2021-02-15)
    lot_20210215_0309_d.fits (30 sec data taken at 20:35:58 on 2021-02-15)
    lot_20210215_0312_d.fits (30 sec data taken at 20:37:16 on 2021-02-15)
    lot_20210215_0315_d.fits (30 sec data taken at 20:38:33 on 2021-02-15)
    lot_20210215_0318_d.fits (30 sec data taken at 20:39:53 on 2021-02-15)
    lot_20210215_0321_d.fits (30 sec data taken at 20:41:12 on 2021-02-15)
    lot_20210215_0324_d.fits (30 sec data taken at 20:42:31 on 2021-02-15)
    lot_20210215_0327_d.fits (30 sec data taken at 20:43:48 on 2021-02-15)
    lot_20210215_0330_d.fits (30 sec data taken at 20:45:08 on 2021-02-15)
    lot_20210215_0333_d.fits (30 sec data taken at 20:46:26 on 2021-02-15)
    lot_20210215_0336_d.fits (30 sec data taken at 20:47:44 on 2021-02-15)
    lot_20210215_0339_d.fits (30 sec data taken at 20:49:04 on 2021-02-15)
    lot_20210215_0342_d.fits (30 sec data taken at 20:50:24 on 2021-02-15)
    lot_20210215_0345_d.fits (30 sec data taken at 20:51:43 on 2021-02-15)
    lot_20210215_0348_d.fits (30 sec data taken at 20:53:01 on 2021-02-15)
    lot_20210215_0351_d.fits (30 sec data taken at 20:54:21 on 2021-02-15)
  ip_Astrodon_2019 band flatfield:
    lot_20210215_0414_d.fits (5 sec data taken at 21:07:17 on 2021-02-15)
    lot_20210215_0417_d.fits (5 sec data taken at 21:07:50 on 2021-02-15)
    lot_20210215_0420_d.fits (5 sec data taken at 21:08:19 on 2021-02-15)
    lot_20210215_0423_d.fits (5 sec data taken at 21:08:49 on 2021-02-15)
    lot_20210215_0426_d.fits (5 sec data taken at 21:09:18 on 2021-02-15)
    lot_20210215_0429_d.fits (5 sec data taken at 21:09:47 on 2021-02-15)
    lot_20210215_0432_d.fits (5 sec data taken at 21:10:15 on 2021-02-15)
    lot_20210215_0435_d.fits (5 sec data taken at 21:10:47 on 2021-02-15)
    lot_20210215_0438_d.fits (5 sec data taken at 21:11:17 on 2021-02-15)
    lot_20210215_0441_d.fits (5 sec data taken at 21:11:46 on 2021-02-15)
    lot_20210215_0444_d.fits (5 sec data taken at 21:12:15 on 2021-02-15)
    lot_20210215_0447_d.fits (5 sec data taken at 21:12:44 on 2021-02-15)
    lot_20210215_0450_d.fits (5 sec data taken at 21:13:14 on 2021-02-15)
    lot_20210215_0453_d.fits (5 sec data taken at 21:13:42 on 2021-02-15)
    lot_20210215_0456_d.fits (5 sec data taken at 21:14:12 on 2021-02-15)
    lot_20210215_0459_d.fits (5 sec data taken at 21:14:41 on 2021-02-15)
    lot_20210215_0462_d.fits (5 sec data taken at 21:15:10 on 2021-02-15)
    lot_20210215_0465_d.fits (5 sec data taken at 21:15:39 on 2021-02-15)
    lot_20210215_0468_d.fits (5 sec data taken at 21:16:08 on 2021-02-15)
    lot_20210215_0471_d.fits (5 sec data taken at 21:16:37 on 2021-02-15)
  rp_Astrodon_2019 band flatfield:
    lot_20210215_0354_d.fits (5 sec data taken at 20:56:05 on 2021-02-15)
    lot_20210215_0357_d.fits (5 sec data taken at 20:56:33 on 2021-02-15)
    lot_20210215_0360_d.fits (5 sec data taken at 20:57:03 on 2021-02-15)
    lot_20210215_0363_d.fits (5 sec data taken at 20:57:31 on 2021-02-15)
    lot_20210215_0366_d.fits (5 sec data taken at 20:57:59 on 2021-02-15)
    lot_20210215_0369_d.fits (5 sec data taken at 20:58:28 on 2021-02-15)
    lot_20210215_0372_d.fits (5 sec data taken at 20:58:56 on 2021-02-15)
    lot_20210215_0375_d.fits (5 sec data taken at 20:59:24 on 2021-02-15)
    lot_20210215_0378_d.fits (5 sec data taken at 20:59:52 on 2021-02-15)
    lot_20210215_0381_d.fits (5 sec data taken at 21:00:21 on 2021-02-15)
    lot_20210215_0384_d.fits (5 sec data taken at 21:00:47 on 2021-02-15)
    lot_20210215_0387_d.fits (5 sec data taken at 21:01:14 on 2021-02-15)
    lot_20210215_0390_d.fits (5 sec data taken at 21:01:42 on 2021-02-15)
    lot_20210215_0393_d.fits (5 sec data taken at 21:02:08 on 2021-02-15)
    lot_20210215_0396_d.fits (5 sec data taken at 21:02:35 on 2021-02-15)
    lot_20210215_0399_d.fits (5 sec data taken at 21:03:01 on 2021-02-15)
    lot_20210215_0402_d.fits (5 sec data taken at 21:03:27 on 2021-02-15)
    lot_20210215_0405_d.fits (5 sec data taken at 21:03:56 on 2021-02-15)
    lot_20210215_0408_d.fits (5 sec data taken at 21:04:26 on 2021-02-15)
    lot_20210215_0411_d.fits (5 sec data taken at 21:04:55 on 2021-02-15)
```

6.2 Combining dark-subtracted flatfield frames

Make a Python script to combine dark-subtracted flatfield frames.

Python Code 12: ao2021_s08_12.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing datetime module
import datetime

# importing numpy module
import numpy
import numpy.ma

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

# construction of parser object
desc = 'combining flatfield frames'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
list_datatype  = ['BIAS', 'DARK', 'FLAT', 'LIGHT']
list_rejection = ['NONE', 'sigclip']
```

```
list_cenfunc   = [’mean’, ’median’]
parser.add_argument (’-f’, ’--filter’, default=’’, help=’filter name’)
parser.add_argument (’-d’, ’--datatype’, default=’BIAS’, \
                     choices=list_datatype, help=’data type’)
parser.add_argument (’-r’, ’--rejection’, default=’NONE’, \
                     choices=list_rejection, \
                     help=’rejection algorithm (default: NONE)’)
parser.add_argument (’-t’, ’--threshold’, type=float, default=4.0, \
                     help=’threshold for sigma clipping (default: 4.0)’)
parser.add_argument (’-n’, ’--maxiters’, type=int, default=10, \
                     help=’maximum number of iterations (default: 10)’)
parser.add_argument (’-c’, ’--cenfunc’, choices=list_cenfunc, \
                     default=’median’, \
                     help=’method to estimate centre value (default: median)’)
parser.add_argument (’-o’, ’--output’, default=’’, help=’output file name’)
parser.add_argument (’files’, nargs=’+’, help=’FITS files’)

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

# input parameters
filter0     = args.filter
datatype0   = args.datatype
rejection   = args.rejection
threshold   = args.threshold
maxiters    = args.maxiters
cenfunc     = args.cenfunc
file_output = args.output
list_files  = args.files

# examination of output file name
if (file_output == ’’):
    print ("Output file name must be given.")
    sys.exit ()
if not (file_output[-5:] == ’.fits’):
    print ("Output file must be a FITS file.")
    sys.exit ()

# command name
command = sys.argv[0]

# date/time
now = datetime.datetime.now ().isoformat ()

# declaring an empty dictionary for storing FITS file information
dict_target = {}

# printing information
print ("# Data search condition:")
print ("#   data type = %s" % datatype0)
print ("#   filter    = \"%s\"" % filter0)
print ("# Input parameters")
print ("#   rejection algorithm = %s" % rejection)
print ("#   threshold of sigma-clipping = %f" % threshold)
print ("#   maximum number of iterations = %d" % maxiters)

# printing status
print ("#")
print ("# Now scanning data...")
```

```
# processing FITS files
for file_fits in list_files:
    # if the extension of the file is not ’.fits’, the we skip
    if not (file_fits[-5:] == ’.fits’):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[’IMAGETYP’]
    # exptime
    exptime  = header[’EXPTIME’]
    # date-obs
    date_obs = header[’DATE-OBS’]
    # time-obs
    time_obs = header[’TIME-OBS’]

    # if the data type is not "FLAT", the we skip the file
    if not (datatype == ’FLAT’):
        continue

    # filter name
    filter_name = header[’FILTER’]

    # appending FITS header information to the dictionary
    if not (filter_name in dict_target):
        dict_target[filter_name] = {}
    dict_target[filter_name][file_fits] = {}
    dict_target[filter_name][file_fits][’exptime’]  = exptime
    dict_target[filter_name][file_fits][’date-obs’] = date_obs
    dict_target[filter_name][file_fits][’time-obs’] = time_obs

# printing status
print ("#")
print ("# Finished scanning files")
print ("#   %d files are found for combining" \
       % len (dict_target[filter0]) )

# checking number of target files
if ( len (dict_target) < 2 ):
    print ("number of target files must be greater than 1.")
    sys.exit ()

print ("#")
print ("# Target files:")
for file_fits in dict_target[filter0]:
    print ("#   %s" % file_fits)

# counter
i = 0
```

```
# list for median pixel values
list_median = []

# printing status
print ("#")
print ("# Reading image data...")

# reading dark frames
for file_fits in dict_target[filter0]:
    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

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

    # image 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)
    list_median.append (median)

    # scaling
    data_scaled = data / median * median_ref

    # constructing a 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 "i"
    i += 1

    # printing status
    print ("#   %04d: \"%s\" (median: %8.2f ADU)" % (i + 1, file_fits, median) )

# printing status
print ("#")
print ("# Finished reading image data")

# printing status
print ("#")
print ("# Combining image...")

# combining flat frames
if (rejection == 'sigclip'):
    # sigma clipping
    clipped_cube = \
        astropy.stats.sigma_clip (cube, sigma=threshold, \
                                  maxiters=maxiters, cenfunc=cenfunc, \
                                  axis=0, masked=True)
    # combining using average
    combined = numpy.ma.average (clipped_cube, weights=list_median, axis=0)
elif (rejection == 'NONE'):
    # combining using average
    combined = numpy.ma.average (cube, weights=list_median, axis=0)

# printing status
print ("#")
print ("# Finished combining image")

# printing status
print ("#")
print ("# Writing image into a new FITS file...")
print ("#    output file = %s" % file_output)

# mean of combined image
mean_combined = numpy.ma.mean (combined)

# adding comments to the header
header['history'] = "FITS file created by the command \"%s\"" % (command)
header['history'] = "Updated on %s" % (now)
header['comment'] = "multiple FITS files are combined into a single FITS file"
header['comment'] = "List of combined files:"
for file_fits in dict_target[filter0]:
    header['comment'] = "  %s" % (file_fits)
header['comment'] = "Options given:"
header['comment'] = "  rejection = %s" % (rejection)
header['comment'] = "  threshold = %f sigma" % (threshold)
header['comment'] = "  maxiters  = %d" % (maxiters)
header['comment'] = "  cenfunc   = %s" % (cenfunc)

# writing a new FITS file
astropy.io.fits.writeto (file_output, \
                         numpy.ma.filled (combined, fill_value=mean_combined), \
                         header=header)

# printing status
print ("#")
print ("# Finished writing image into a new FITS file")
print ("#")
```

Execute the script, and make combined g'-band flatfield.

```
% chmod a+x ao2021_s08_12.py
% ./ao2021_s08_12.py -h
usage: ao2021_s08_12.py [-h] [-f FILTER] [-d {BIAS,DARK,FLAT,LIGHT}]
                        [-r {NONE,sigclip}] [-t THRESHOLD] [-n MAXITERS]
                        [-c {mean,median}] [-o OUTPUT]
                        files [files ...]

combining flatfield frames

positional arguments:
  files                 FITS files
```

```
optional arguments:
  -h, --help            show this help message and exit
  -f FILTER, --filter FILTER
                        filter name
  -d {BIAS,DARK,FLAT,LIGHT}, --datatype {BIAS,DARK,FLAT,LIGHT}
                        data type
  -r {NONE,sigclip}, --rejection {NONE,sigclip}
                        rejection algorithm (default: NONE)
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for sigma clipping (default: 4.0)
  -n MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 10)
  -c {mean,median}, --cenfunc {mean,median}
                        method to estimate centre value (default: median)
  -o OUTPUT, --output OUTPUT
                        output file name

% ./ao2021_s08_12.py -f gp_Astrodon_2019 -d FLAT -r sigclip \
? -o flat_gp_Astrodon_2019.fits *_d.fits
# Data search condition:
#   data type = FLAT
#   filter    = "gp_Astrodon_2019"
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# Now scanning data...
#
# Finished scanning files
#   20 files are found for combining
#
# Target files:
#   lot_20210215_0294_d.fits
#   lot_20210215_0297_d.fits
#   lot_20210215_0300_d.fits
#   lot_20210215_0303_d.fits
#   lot_20210215_0306_d.fits
#   lot_20210215_0309_d.fits
#   lot_20210215_0312_d.fits
#   lot_20210215_0315_d.fits
#   lot_20210215_0318_d.fits
#   lot_20210215_0321_d.fits
#   lot_20210215_0324_d.fits
#   lot_20210215_0327_d.fits
#   lot_20210215_0330_d.fits
#   lot_20210215_0333_d.fits
#   lot_20210215_0336_d.fits
#   lot_20210215_0339_d.fits
#   lot_20210215_0342_d.fits
#   lot_20210215_0345_d.fits
#   lot_20210215_0348_d.fits
#   lot_20210215_0351_d.fits
#
# Reading image data...
#   0002: "lot_20210215_0294_d.fits" (median: 12897.10 ADU)
#   0003: "lot_20210215_0297_d.fits" (median: 13235.57 ADU)
#   0004: "lot_20210215_0300_d.fits" (median: 12899.20 ADU)
#   0005: "lot_20210215_0303_d.fits" (median: 12853.87 ADU)
#   0006: "lot_20210215_0306_d.fits" (median: 13398.80 ADU)
#   0007: "lot_20210215_0309_d.fits" (median: 12698.17 ADU)
#   0008: "lot_20210215_0312_d.fits" (median: 13034.23 ADU)
#   0009: "lot_20210215_0315_d.fits" (median: 12188.90 ADU)
#   0010: "lot_20210215_0318_d.fits" (median: 11949.13 ADU)
#   0011: "lot_20210215_0321_d.fits" (median: 12894.33 ADU)
#   0012: "lot_20210215_0324_d.fits" (median: 12467.20 ADU)
#   0013: "lot_20210215_0327_d.fits" (median: 13390.03 ADU)
#   0014: "lot_20210215_0330_d.fits" (median: 14960.73 ADU)
#   0015: "lot_20210215_0333_d.fits" (median: 15291.67 ADU)
#   0016: "lot_20210215_0336_d.fits" (median: 14117.03 ADU)
#   0017: "lot_20210215_0339_d.fits" (median: 15336.13 ADU)
#   0018: "lot_20210215_0342_d.fits" (median: 14869.47 ADU)
#   0019: "lot_20210215_0345_d.fits" (median: 15123.87 ADU)
#   0020: "lot_20210215_0348_d.fits" (median: 13524.50 ADU)
#   0021: "lot_20210215_0351_d.fits" (median: 14025.57 ADU)
#
# Finished reading image data
#
# Combining image...
#
# Finished combining image
#
# Writing image into a new FITS file...
#   output file = flat_gp_Astrodon_2019.fits
#
# Finished writing image into a new FITS file
#
% ls -l flat_gp_Astrodon_2019.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:01 flat_gp_Astrodon_2019.fits
```

Make combined r'-band flatfield.

```
% ./ao2021_s08_12.py -f rp_Astrodon_2019 -d FLAT -r sigclip \
? -o flat_rp_Astrodon_2019.fits *_d.fits
# Data search condition:
#   data type = FLAT
#   filter    = "rp_Astrodon_2019"
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# Now scanning data...
#
# Finished scanning files
#   20 files are found for combining
#
# Target files:
#   lot_20210215_0354_d.fits
#   lot_20210215_0357_d.fits
#   lot_20210215_0360_d.fits
#   lot_20210215_0363_d.fits
#   lot_20210215_0366_d.fits
#   lot_20210215_0369_d.fits
#   lot_20210215_0372_d.fits
#   lot_20210215_0375_d.fits
#   lot_20210215_0378_d.fits
#   lot_20210215_0381_d.fits
#   lot_20210215_0384_d.fits
#   lot_20210215_0387_d.fits
#   lot_20210215_0390_d.fits
#   lot_20210215_0393_d.fits
#   lot_20210215_0396_d.fits
#   lot_20210215_0399_d.fits
#   lot_20210215_0402_d.fits
#   lot_20210215_0405_d.fits
#   lot_20210215_0408_d.fits
#   lot_20210215_0411_d.fits
#
# Reading image data...
#   0002: "lot_20210215_0354_d.fits" (median: 18973.36 ADU)
#   0003: "lot_20210215_0357_d.fits" (median: 19151.26 ADU)
#   0004: "lot_20210215_0360_d.fits" (median: 17531.36 ADU)
#   0005: "lot_20210215_0363_d.fits" (median: 18534.14 ADU)
#   0006: "lot_20210215_0366_d.fits" (median: 18035.62 ADU)
#   0007: "lot_20210215_0369_d.fits" (median: 18493.10 ADU)
#   0008: "lot_20210215_0372_d.fits" (median: 18591.22 ADU)
#   0009: "lot_20210215_0375_d.fits" (median: 17195.68 ADU)
#   0010: "lot_20210215_0378_d.fits" (median: 18695.28 ADU)
#   0011: "lot_20210215_0381_d.fits" (median: 18222.34 ADU)
#   0012: "lot_20210215_0384_d.fits" (median: 17885.28 ADU)
#   0013: "lot_20210215_0387_d.fits" (median: 18918.22 ADU)
#   0014: "lot_20210215_0390_d.fits" (median: 18598.78 ADU)
#   0015: "lot_20210215_0393_d.fits" (median: 18282.40 ADU)
#   0016: "lot_20210215_0396_d.fits" (median: 18903.86 ADU)
#   0017: "lot_20210215_0399_d.fits" (median: 19064.10 ADU)
#   0018: "lot_20210215_0402_d.fits" (median: 19267.10 ADU)
#   0019: "lot_20210215_0405_d.fits" (median: 19029.04 ADU)
#   0020: "lot_20210215_0408_d.fits" (median: 18909.50 ADU)
#   0021: "lot_20210215_0411_d.fits" (median: 18692.44 ADU)
#
# Finished reading image data
#
# Combining image...
#
# Finished combining image
#
# Writing image into a new FITS file...
#   output file = flat_rp_Astrodon_2019.fits
#
# Finished writing image into a new FITS file
#
% ls -l flat_*
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:01 flat_gp_Astrodon_2019.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:05 flat_rp_Astrodon_2019.fits
```

Make combined i'-band flatfield.

```
% ./ao2021_s08_12.py -f ip_Astrodon_2019 -d FLAT -r sigclip \
? -o flat_ip_Astrodon_2019.fits *_d.fits
# Data search condition:
#   data type = FLAT
#   filter    = "ip_Astrodon_2019"
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# Now scanning data...
#
# Finished scanning files
#   20 files are found for combining
#
# Target files:
#   lot_20210215_0414_d.fits
#   lot_20210215_0417_d.fits
#   lot_20210215_0420_d.fits
#   lot_20210215_0423_d.fits
#   lot_20210215_0426_d.fits
#   lot_20210215_0429_d.fits
#   lot_20210215_0432_d.fits
#   lot_20210215_0435_d.fits
#   lot_20210215_0438_d.fits
#   lot_20210215_0441_d.fits
#   lot_20210215_0444_d.fits
#   lot_20210215_0447_d.fits
#   lot_20210215_0450_d.fits
#   lot_20210215_0453_d.fits
#   lot_20210215_0456_d.fits
#   lot_20210215_0459_d.fits
#   lot_20210215_0462_d.fits
#   lot_20210215_0465_d.fits
#   lot_20210215_0468_d.fits
#   lot_20210215_0471_d.fits
#
# Reading image data...
#   0002: "lot_20210215_0414_d.fits" (median: 19774.82 ADU)
#   0003: "lot_20210215_0417_d.fits" (median: 19404.79 ADU)
#   0004: "lot_20210215_0420_d.fits" (median: 19067.08 ADU)
#   0005: "lot_20210215_0423_d.fits" (median: 17902.12 ADU)
#   0006: "lot_20210215_0426_d.fits" (median: 17827.04 ADU)
#   0007: "lot_20210215_0429_d.fits" (median: 18017.06 ADU)
#   0008: "lot_20210215_0432_d.fits" (median: 17922.72 ADU)
#   0009: "lot_20210215_0435_d.fits" (median: 17942.78 ADU)
#   0010: "lot_20210215_0438_d.fits" (median: 16950.18 ADU)
#   0011: "lot_20210215_0441_d.fits" (median: 17164.66 ADU)
#   0012: "lot_20210215_0444_d.fits" (median: 17656.26 ADU)
#   0013: "lot_20210215_0447_d.fits" (median: 17861.22 ADU)
#   0014: "lot_20210215_0450_d.fits" (median: 17714.04 ADU)
#   0015: "lot_20210215_0453_d.fits" (median: 17913.54 ADU)
#   0016: "lot_20210215_0456_d.fits" (median: 17827.86 ADU)
#   0017: "lot_20210215_0459_d.fits" (median: 18337.40 ADU)
#   0018: "lot_20210215_0462_d.fits" (median: 18101.64 ADU)
#   0019: "lot_20210215_0465_d.fits" (median: 18468.18 ADU)
#   0020: "lot_20210215_0468_d.fits" (median: 17987.86 ADU)
#   0021: "lot_20210215_0471_d.fits" (median: 19208.12 ADU)
#
# Finished reading image data
#
# Combining image...
#
# Finished combining image
#
```

```
# Writing image into a new FITS file...
#   output file = flat_ip_Astrodon_2019.fits
#
# Finished writing image into a new FITS file
#
% ls -l flat_*
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:01 flat_gp_Astrodon_2019.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:07 flat_ip_Astrodon_2019.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:05 flat_rp_Astrodon_2019.fits
```

6.3 Normalising combined flatfield

Make a Python script to normalise combined flatfield.

Python Code 13: ao2021_s08_13.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = ’normalising FITS file’
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument (’-o’, ’--output’, default=’’, help=’output file name’)
parser.add_argument (’file_input’, nargs=1, help=’input FITS file’)

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

# input parameters
file_output = args.output
file_input  = args.file_input[0]

# checking input file name
if not (file_input[-5:] == ’.fits’):
    print ("Input file must be a FITS file.")
    sys.exit ()

# checking output file name
if (file_output == ’’):
    print ("Output file name has to be given.")
    sys.exit ()
if not (file_output[-5:] == ’.fits’):
    print ("Output file must be a FITS file.")
    sys.exit ()

# command name
command = sys.argv[0]

# date/time
now = datetime.datetime.now ().isoformat ()

# printing status
print ("#")
print ("# input file  = %s" % file_input)
print ("# output file = %s" % file_output)
print ("#")

# opening FITS file
hdu_list = astropy.io.fits.open (file_input)

# header of primary HDU
header = hdu_list[0].header

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

# closing FITS file
hdu_list.close ()

# mean of pixel values
mean = numpy.mean (data)

# printing status
print ("# mean value of input file = %f" % mean)

# normalisation
data_normalised = data / mean

# adding comments to the header
header['history'] = "FITS file created by the command \"%s\"" % (command)
header['history'] = "Updated on %s" % (now)
header['comment'] = "normalisation of a FITS file"
header['comment'] = "Input file: %s" % (file_input)
header['comment'] = "Mean of pixel values of input file = %f" % (mean)

# writing a new FITS file
astropy.io.fits.writeto (file_output, data_normalised, header=header)
```

Run the script, and normalise g'-band flatfield.

```
% chmod a+x ao2021_s08_13.py
% ./ao2021_s08_13.py -h
usage: ao2021_s08_13.py [-h] [-o OUTPUT] file_input

normalising FITS file

positional arguments:
  file_input            input FITS file

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        output file name

% ./ao2021_s08_13.py -o nflat_gp_Astrodon_2019.fits flat_gp_Astrodon_2019.fits
#
# input file  = flat_gp_Astrodon_2019.fits
# output file = nflat_gp_Astrodon_2019.fits
#
# mean value of input file = 12885.979308
% ls -l nflat_gp_Astrodon_2019.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:45 nflat_gp_Astrodon_2019.fits
```

Normalise r'-band flatfield.

```
% ./ao2021_s08_13.py -o nflat_rp_Astrodon_2019.fits flat_rp_Astrodon_2019.fits
#
# input file  = flat_rp_Astrodon_2019.fits
# output file = nflat_rp_Astrodon_2019.fits
#
# mean value of input file = 18961.987258
% ls -l nflat_*.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:45 nflat_gp_Astrodon_2019.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:47 nflat_rp_Astrodon_2019.fits
```

Normalise i'-band flatfield.

```
% ./ao2021_s08_13.py -o nflat_ip_Astrodon_2019.fits flat_ip_Astrodon_2019.fits
#
# input file  = flat_ip_Astrodon_2019.fits
# output file = nflat_ip_Astrodon_2019.fits
#
# mean value of input file = 19747.061215
% ls -l nflat_*.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:45 nflat_gp_Astrodon_2019.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:48 nflat_ip_Astrodon_2019.fits
-rw-r--r--  1 daisuke  taiwan  33563520 Apr  5 23:47 nflat_rp_Astrodon_2019.fits
```

6.4 Checking mean value of normalised flatfield

Make a Python script to check mean value of normalised flatfield.

Python Code 14: ao2021_s08_14.py

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

# importing argparse module
import argparse

# importing numpy module
import numpy
import numpy.ma

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

```
# construction pf parser object
desc = ’calculating statistical information of FITS files’
parser = argparse.ArgumentParser (description=desc)

# adding arguments
list_rejection = [’NONE’, ’sigclip’]
parser.add_argument (’-r’, ’--rejection’, default=’NONE’, \
                     choices=list_rejection, help=’rejection algorithm’)
parser.add_argument (’-t’, ’--threshold’, type=float, default=4.0, \
                     help=’threshold for sigma clipping’)
parser.add_argument (’-n’, ’--maxiters’, type=int, default=10, \
                     help=’maximum number of iterations’)
parser.add_argument (’files’, nargs=’+’, help=’FITS files’)

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

# input parameters
rejection  = args.rejection
threshold  = args.threshold
maxiters   = args.maxiters
list_files = args.files

# printing information
print ("# Input parameters")
print ("#   rejection algorithm = %s" % rejection)
print ("#   threshold of sigma-clipping = %f" % threshold)
print ("#   maximum number of iterations = %d" % maxiters)

# printing header
print ("#")
print ("# %-25s %8s %8s %8s %7s %8s %8s" \
       % (’file name’, ’npix’, ’mean’, ’median’, ’stddev’, ’min’, ’max’) )
print ("#")

# scanning files
for file_fits in list_files:
    # if the file is not a FITS file, then skip
    if not (file_fits[-5:] == ’.fits’):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # image of primary HDU
    data0 = hdu_list[0].data

    # closing FITS file
    hdu_list.close ()
    # calculations

    # for no rejection algorithm
    if (rejection == 'NONE'):
        # making a masked array
        data1 = numpy.ma.array (data0, mask=False)
    # for sigma clipping algorithm
    elif (rejection == 'sigclip'):
        data1 = numpy.ma.array (data0, mask=False)
        # iterations
        for j in range (maxiters):
            # number of usable pixels of previous iterations
            npix_prev = len (numpy.ma.compressed (data1) )
            # calculation of median
            median = numpy.ma.median (data1)
            # calculation of standard deviation
            stddev = numpy.ma.std (data1)
            # lower threshold
            low = median - threshold * stddev
            # higher threshold
            high = median + threshold * stddev
            # making a mask
            mask = (data1 < low) | (data1 > high)
            # making a masked array
            data1 = numpy.ma.array (data0, mask=mask)
            # number of usable pixels
            npix_now = len (numpy.ma.compressed (data1) )
            # leaving the loop, if number of usable pixels do not change
            if (npix_now == npix_prev):
                break

    # calculation of mean, median, stddev, min, and max
    mean   = numpy.ma.mean (data1)
    median = numpy.ma.median (data1)
    stddev = numpy.ma.std (data1)
    vmin   = numpy.ma.min (data1)
    vmax   = numpy.ma.max (data1)

    # number of pixels
    npix = len (data1.compressed () )

    # file name
    filename = file_fits.split ('/') [-1]

    # printing result
    print ("%-27s %8d %8.2f %8.2f %7.2f %8.2f %8.2f" \
           % (filename, npix, mean, median, stddev, vmin, vmax) )
```

Execute the script.

```
% chmod a+x ao2021_s08_14.py
% ./ao2021_s08_14.py -h
usage: ao2021_s08_14.py [-h] [-r {NONE,sigclip}] [-t THRESHOLD] [-n MAXITERS]
                        files [files ...]

calculating statistical information of FITS files
```

```
positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -r {NONE,sigclip}, --rejection {NONE,sigclip}
                        rejection algorithm
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for sigma clipping
  -n MAXITERS, --maxiters MAXITERS
                        maximum number of iterations

% ./ao2021_s08_14.py *flat*.fits
# Input parameters
#   rejection algorithm = NONE
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# file name                     npix     mean   median  stddev      min      max
#
flat_gp_Astrodon_2019.fits   4194304 12885.98 12897.81  277.53  6320.95 13764.87
flat_ip_Astrodon_2019.fits   4194304 19747.06 19774.43  665.90  9492.93 21378.68
flat_rp_Astrodon_2019.fits   4194304 18961.99 18973.91  470.09  8966.51 20267.85
nflat_gp_Astrodon_2019.fits  4194304     1.00     1.00    0.02     0.49     1.07
nflat_ip_Astrodon_2019.fits  4194304     1.00     1.00    0.03     0.48     1.08
nflat_rp_Astrodon_2019.fits  4194304     1.00     1.00    0.02     0.47     1.07
```

Normalised flatfield frames has mean values of unity.

6.5 Visual inspection of normalised flatfield

Use Ginga to show the image of g'-band normalise flatfield. (Fig. 5)

```
% ginga nflat_gp_Astrodon_2019.fits
```

Also, check r'-band and i'-band normalised flatfield.

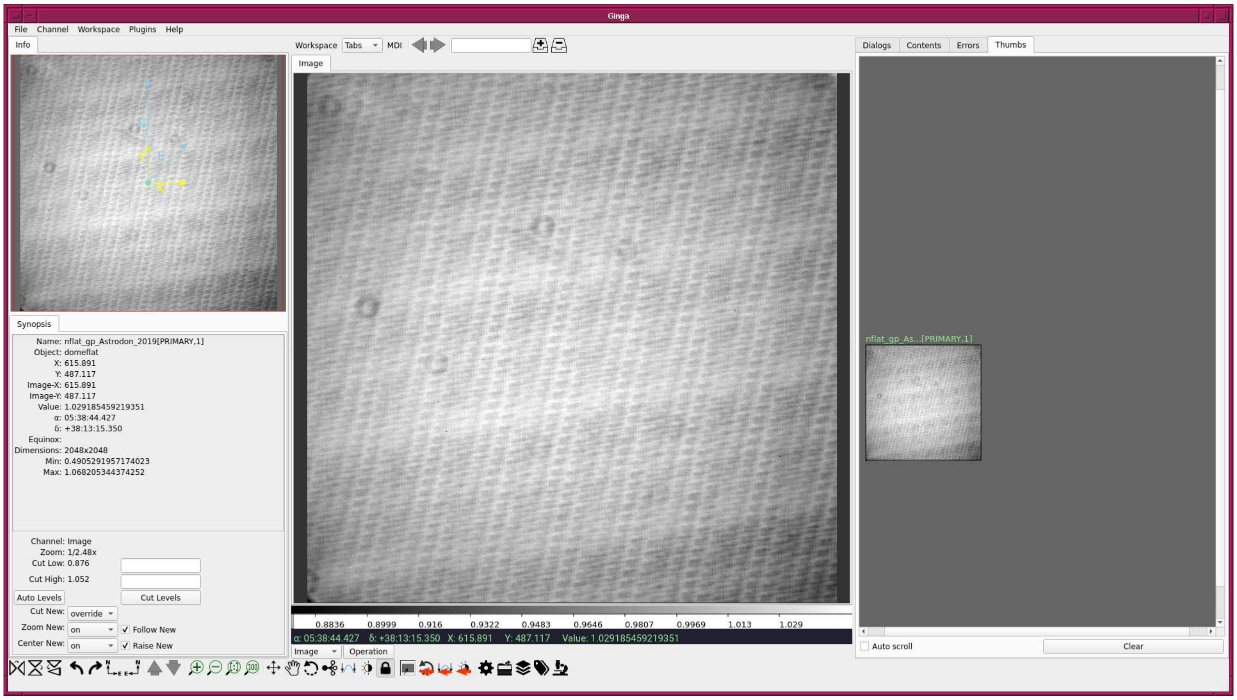
7 Flatfielding

7.1 Listing dark-subtracted object frames

Make a Python script to list dark subtracted object frames.

Python Code 15: ao2021_s08_15.py

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

# importing argparse module
import argparse

# importing pathlib module
import pathlib

# importing sys module
import sys

# importing astropy module
```



Name: nflat_gp_Astrodon_2019[PRIMARY,1]
Object: domeflat
X: 615.891
Y: 487.117
Image-X: 615.891
Image-Y: 487.117
Value: 1.029185459219351
α: 05:38:44.427
δ: +38:13:15.350
Equinox:
Dimensions: 2048x2048
Min: 0.4905291957174023
Max: 1.068205344374252

Channel: Image
Zoom: 1/2.48x
Cut Low: 0.876
Cut High: 1.052
Cut New: override
Zoom New: on ☑ Follow New
Center New: on ☑ Raise New

α: 05:38:44.427 δ: +38:13:15.350 X: 615.891 Y: 487.117 Value: 1.029185459219351

Figure 5: The image of g'-band normalised flatfield.

```
import astropy.io.fits

# construction of parser object
desc = 'listing dark subtracted object frames'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
default_filter_keyword   = 'FILTER'
default_datatype_keyword = 'IMAGETYP'
default_exptime_keyword  = 'EXPTIME'
parser.add_argument ('-f', '--filter', default=default_filter_keyword, \
                     help='FITS keyword for filter name')
parser.add_argument ('-d', '--datatype', default=default_datatype_keyword, \
                     help='FITS keyword for data type')
parser.add_argument ('-e', '--exptime', default=default_exptime_keyword, \
                     help='FITS keyword for exposure time')
parser.add_argument ('files', nargs='+', help='FITS files')

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

# input parameters
filter_keyword   = args.filter
datatype_keyword = args.datatype
exptime_keyword  = args.exptime
files            = args.files

# declaring an empty dictionary for storing FITS file information
dict_target = {}

# processing FITS files
for file_fits in files:
    # if the extension of the file is not '.fits', the we skip
    if not (file_fits[-5:] == '.fits'):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_fits)

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[datatype_keyword]
    # exptime
    exptime = header[exptime_keyword]
    # date-obs
    date_obs = header['DATE-OBS']
    # time-obs
    time_obs = header['TIME-OBS']

    # if the data type is not "LIGHT", then we skip the file
    if not (datatype == 'LIGHT'):
        continue

    # if the file name is not "*_d.fits", then we skip the file
    if not (file_fits[-7:] == '_d.fits'):
        continue

    # filter name
    filter_name = header[filter_keyword]

    # appending FITS header information to the dictionary
    if not (filter_name in dict_target):
        dict_target[filter_name] = {}
    dict_target[filter_name][file_fits] = {}
    dict_target[filter_name][file_fits]['exptime']  = exptime
    dict_target[filter_name][file_fits]['date-obs'] = date_obs
    dict_target[filter_name][file_fits]['time-obs'] = time_obs

# printing FITS file list
print ("List of FITS files for flatfielding:")
for filter_name in sorted (dict_target.keys () ):
    nflat = "nflat_%s.fits" % filter_name
    path_nflat = pathlib.Path (nflat)
    if not (path_nflat.exists () ):
        print ("The flatfield file \"%s\" does not exist." % nflat)
        print ("Check the data!")
        sys.exit ()
    print ("  %s band data:"% filter_name)
    print ("    normalised flatfield = %s" % nflat)
    for file_fits in sorted (dict_target[filter_name].keys () ):
        flatfielded = file_fits[:-5] + "f.fits"
        print ("      %s / %s" % (file_fits, nflat) )
        print ("          ==> %s" % flatfielded)
print ("Total number of files to be processed:")
total_files = 0
for filter_name in sorted (dict_target.keys () ):
    n_files = len (dict_target[filter_name].keys () )
    print ("  %s band data: %d files" % (filter_name, n_files) )
    total_files += n_files
print ("  Total number of files: %d files" % total_files)
```

Run the script.

```
% chmod a+x ao2021_s08_15.py
% ./ao2021_s08_15.py *.fits
List of FITS files for flatfielding:
  gp_Astrodon_2019 band data:
    normalised flatfield = nflat_gp_Astrodon_2019.fits
      lot_20210215_0114_d.fits / nflat_gp_Astrodon_2019.fits
          ==> lot_20210215_0114_df.fits
      lot_20210215_0115_d.fits / nflat_gp_Astrodon_2019.fits
          ==> lot_20210215_0115_df.fits
      lot_20210215_0136_d.fits / nflat_gp_Astrodon_2019.fits
          ==> lot_20210215_0136_df.fits
      lot_20210215_0137_d.fits / nflat_gp_Astrodon_2019.fits
          ==> lot_20210215_0137_df.fits
      lot_20210215_0158_d.fits / nflat_gp_Astrodon_2019.fits
          ==> lot_20210215_0158_df.fits

.....

      lot_20210215_0289_d.fits / nflat_rp_Astrodon_2019.fits
          ==> lot_20210215_0289_df.fits
      lot_20210215_0290_d.fits / nflat_rp_Astrodon_2019.fits
          ==> lot_20210215_0290_df.fits
      lot_20210215_0291_d.fits / nflat_rp_Astrodon_2019.fits
          ==> lot_20210215_0291_df.fits
      lot_20210215_0292_d.fits / nflat_rp_Astrodon_2019.fits
          ==> lot_20210215_0292_df.fits
      lot_20210215_0293_d.fits / nflat_rp_Astrodon_2019.fits
          ==> lot_20210215_0293_df.fits
Total number of files to be processed:
  gp_Astrodon_2019 band data: 19 files
  ip_Astrodon_2019 band data: 19 files
  rp_Astrodon_2019 band data: 126 files
  Total number of files: 164 files
```

7.2 Carrying out flatfielding

Make a Python script to carry out flatfielding.

Python Code 16: ao2021_s08_16.py

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

# importing argparse module
import argparse

# importing pathlib module
import pathlib

# importing sys module
import sys

# importing datetime module
```

```
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = ’carrying out flatfielding’
parser = argparse.ArgumentParser (description=desc)

# adding arguments
default_filter_keyword   = ’FILTER’
default_datatype_keyword = ’IMAGETYP’
default_exptime_keyword  = ’EXPTIME’
parser.add_argument (’-f’, ’--filter’, default=default_filter_keyword, \
                     help=’FITS keyword for filter name’)
parser.add_argument (’-d’, ’--datatype’, default=default_datatype_keyword, \
                     help=’FITS keyword for data type’)
parser.add_argument (’-e’, ’--exptime’, default=default_exptime_keyword, \
                     help=’FITS keyword for exposure time’)
parser.add_argument (’files’, nargs=’+’, help=’FITS files’)

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

# input parameters
filter_keyword   = args.filter
datatype_keyword = args.datatype
exptime_keyword  = args.exptime
files            = args.files

# command name
command = sys.argv[0]

# date/time
now = datetime.datetime.now ().isoformat ()

# declaring an empty dictionary for storing FITS file information
dict_target = {}

# processing FITS files
for file_darksub in files:
    # if the extension of the file is not ’.fits’, the we skip
    if not (file_darksub[-5:] == ’.fits’):
        continue

    # opening FITS file
    hdu_list = astropy.io.fits.open (file_darksub)

    # header of primary HDU
    header = hdu_list[0].header

    # closing FITS file
    hdu_list.close ()

    # data type
    datatype = header[datatype_keyword]
    # exptime
    exptime = header[exptime_keyword]

    # if the data type is not "LIGHT", then we skip the file
    if not (datatype == 'LIGHT'):
        continue

    # if the file name is not "*_d.fits", then we skip the file
    if not (file_darksub[-7:] == '_d.fits'):
        continue

    # filter name
    filter_name = header[filter_keyword]

    # appending file name to the dictionary
    dict_target[file_darksub] = {}
    dict_target[file_darksub]['filter'] = filter_name
    dict_target[file_darksub]['exptime'] = exptime

# dark subtraction

print ("#")
print ("# Processing each FITS file...")
print ("#")

# processing each FITS file
for file_darksub in sorted (dict_target.keys () ):
    # file names
    file_flatfielded = file_darksub.split ('/') [-1] [:-5] + 'f.fits'
    file_nflat = 'nflat_' + dict_target[file_darksub]['filter'] + '.fits'

    # if normalised flatfield does not exist , then stop the script
    path_nflat = pathlib.Path (file_nflat)
    if not (path_nflat.exists () ):
        print ("The flatfield file \"%s\" does not exist." % file_nflat)
        print ("Check the data!")
        sys.exit ()

    print ("# dividing %s by %s..." % (file_darksub , file_nflat) )
    print ("#   %s ==> %s" % (file_darksub , file_flatfielded) )

    # opening FITS file (raw data)
    hdu_list = astropy.io.fits.open (file_darksub)

    # header of primary HDU
    header = hdu_list[0].header

    # printing status
    print ("#     reading dark-subtracted data from \"%s\"..." % file_darksub)

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

    # closing FITS file
    hdu_list.close ()

    # opening FITS file (nflat)
    hdu_list = astropy.io.fits.open (file_nflat)
    # header of primary HDU
    header_nflat = hdu_list[0].header

    # printing status
    print ("#     reading normalised flatfield data from \"%s\"..." \
           % file_nflat)

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

    # closing FITS file
    hdu_list.close ()

    # printing status
    print ("#     dividing \"%s\" by \"%s\"..." % (file_darksub, file_nflat) )

    # flatfielding
    data_flatfielded = data_darksub / data_nflat

    # adding comments to new FITS file
    header['history'] = "FITS file created by the command \"%s\"" % (command)
    header['history'] = "Updated on %s" % (now)
    header['comment'] = "flatfielding was carried out"
    header['comment'] = "dark-subtracted data: %s" % (file_darksub)
    header['comment'] = "normalised flatfield data: %s" % (file_nflat)
    header['comment'] = "flatfielded data: %s" % (file_flatfielded)

    # printing status
    print ("#     writing new file \"%s\"..." % file_flatfielded)

    # writing a new FITS file
    astropy.io.fits.writeto (file_flatfielded, data_flatfielded, header=header)
```

Execute the script.

```
% chmod a+x ao2021_s08_16.py
% ./ao2021_s08_16.py -h
usage: ao2021_s08_16.py [-h] [-f FILTER] [-d DATATYPE] [-e EXPTIME]
                        files [files ...]

carrying out flatfielding

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -f FILTER, --filter FILTER
                        FITS keyword for filter name
  -d DATATYPE, --datatype DATATYPE
                        FITS keyword for data type
  -e EXPTIME, --exptime EXPTIME
                        FITS keyword for exposure time

% ./ao2021_s08_16.py *.fits
#
```

```
# Processing each FITS file...
#
# dividing lot_20210215_0070_d.fits by nflat_rp_Astrodon_2019.fits...
#   lot_20210215_0070_d.fits ==> lot_20210215_0070_df.fits
#     reading dark-subtracted data from "lot_20210215_0070_d.fits"...
#     reading normalised flatfield data from "nflat_rp_Astrodon_2019.fits"...
#     dividing "lot_20210215_0070_d.fits" by "nflat_rp_Astrodon_2019.fits"...
#     writing new file "lot_20210215_0070_df.fits"...
# dividing lot_20210215_0071_d.fits by nflat_rp_Astrodon_2019.fits...
#   lot_20210215_0071_d.fits ==> lot_20210215_0071_df.fits
#     reading dark-subtracted data from "lot_20210215_0071_d.fits"...
#     reading normalised flatfield data from "nflat_rp_Astrodon_2019.fits"...
#     dividing "lot_20210215_0071_d.fits" by "nflat_rp_Astrodon_2019.fits"...
#     writing new file "lot_20210215_0071_df.fits"...

.....

# dividing lot_20210215_0292_d.fits by nflat_rp_Astrodon_2019.fits...
#   lot_20210215_0292_d.fits ==> lot_20210215_0292_df.fits
#     reading dark-subtracted data from "lot_20210215_0292_d.fits"...
#     reading normalised flatfield data from "nflat_rp_Astrodon_2019.fits"...
#     dividing "lot_20210215_0292_d.fits" by "nflat_rp_Astrodon_2019.fits"...
#     writing new file "lot_20210215_0292_df.fits"...
# dividing lot_20210215_0293_d.fits by nflat_rp_Astrodon_2019.fits...
#   lot_20210215_0293_d.fits ==> lot_20210215_0293_df.fits
#     reading dark-subtracted data from "lot_20210215_0293_d.fits"...
#     reading normalised flatfield data from "nflat_rp_Astrodon_2019.fits"...
#     dividing "lot_20210215_0293_d.fits" by "nflat_rp_Astrodon_2019.fits"...
#     writing new file "lot_20210215_0293_df.fits"...
```

7.3 Visual inspection of flatfielded data

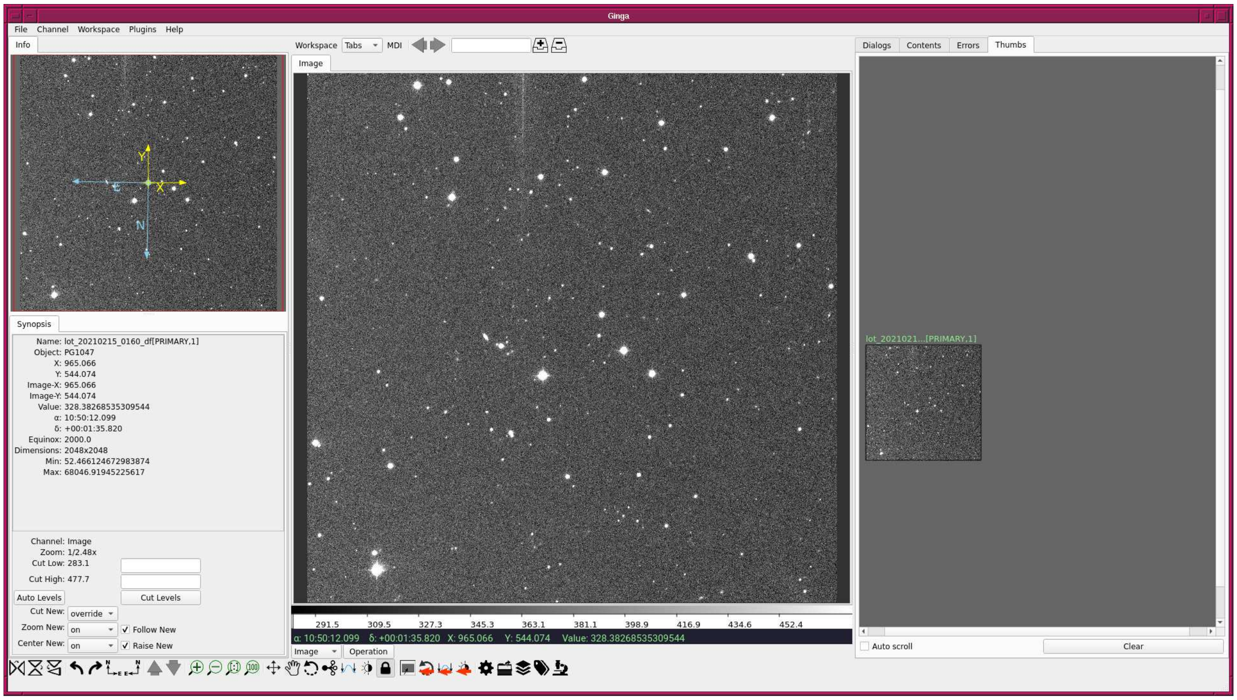
Use Ginga to show the image of flatfieded data. (Fig. 6)

```
% ginga lot_20210215_0160_df.fits
```

7.4 Blinking images

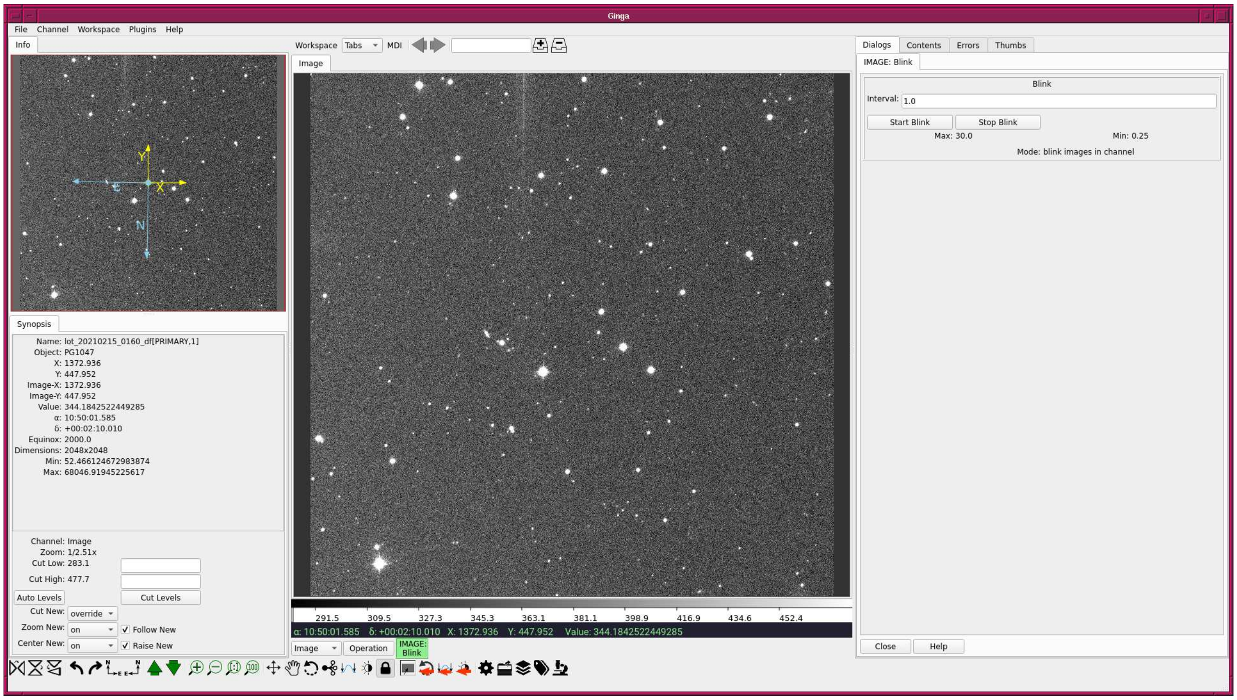
Compare raw data, dark-subtracted data, and flatfielded data using blinking. Try following.

1. Start Ginga.
2. Load the image `data_ao2021_s08/lot_20210215_0160.fits`.
3. Load the iamge `lot_20210215_0160_d.fits`.
4. Load the iamge `lot_20210215_0160_df.fits`.
5. Click the menu “Plugins”.
6. Select the menu “Blink Images”.
7. Push the button “Start Blink”. (Fig. 7)



```
        Name: lot_20210215_0160_df[PRIMARY,1]
      Object: PG1047
           X: 965.066
           Y: 544.074
     Image-X: 965.066
     Image-Y: 544.074
       Value: 328.38268535309544
           α: 10:50:12.099
           δ: +00:01:35.820
     Equinox: 2000.0
  Dimensions: 2048x2048
         Min: 52.466124672983874
         Max: 68046.91945225617

     Channel: Image
        Zoom: 1/2.48x
     Cut Low: 283.1
    Cut High: 477.7
     Cut New: override
    Zoom New: on        ☑ Follow New
  Center New: on        ☑ Raise New

α: 10:50:12.099   δ: +00:01:35.820   X: 965.066   Y: 544.074   Value: 328.38268535309544
```

Figure 6: An example of flatfielded data.



```
        Name: lot_20210215_0160_df[PRIMARY,1]
      Object: PG1047
           X: 1372.936
           Y: 447.952
     Image-X: 1372.936
     Image-Y: 447.952
       Value: 344.1842522449285
           α: 10:50:01.585
           δ: +00:02:10.010
     Equinox: 2000.0
  Dimensions: 2048x2048
         Min: 52.466124672983874
         Max: 68046.91945225617

     Channel: Image
        Zoom: 1/2.51x
     Cut Low: 283.1
    Cut High: 477.7
     Cut New: override
    Zoom New: on        ☑ Follow New
  Center New: on        ☑ Raise New

α: 10:50:01.585   δ: +00:02:10.010   X: 1372.936   Y: 447.952   Value: 344.1842522449285

Interval: 1.0
Max: 30.0          Min: 0.25
Mode: blink images in channel
```

Figure 7: Blinking images using Ginga.

8 For your training

- 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`
- 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`
- 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`

9 Assignment

1. Describe basic CCD data reduction for imaging data. What do we need to do? Why do we need to subtract a dark frame from an object frame? Why do we need to divide the object frame by a flatfield frame?

2. Make three useful utility programs for CCD data reduction written by Python. Describe the functions of your utility programs. Show the source codes of your scripts. Execute the script, take screenshots of your computer display, and show the behaviour of your scripts.

3. Make a single Python script to scan the FITS files obtained from a single night observation and examine whether or not the data set is self-consistent. Describe what are needed to be checked. Explain the design of your Python script. Use a flow chart or a block diagram to explain the design of your Python script. Show the source code of your Python script. Run the script, take a screenshot of your computer display, and show the result. Prepare a set of data lack of some flatfield (or dark) frames, and run the script.

4. Make a single Python script to carry out followings.

    (a) combining dark frames,

    (b) dark subtraction,

    (c) combining flatfield frames,

    (d) flatfielding.

    Explain the design of your Python script. Use a flow chart or a block diagram to explain the design of your Python script. Show the source code of your Python script. Run the script, and check the behaviour of the script.

5. Make a single Python script to work as a data reduction pipeline to check the self-consistency of the data and carry out basic CCD data reduction. Explain the design of your Python script. Use a flow chart or a block diagram to explain the design of your Python script. Show the source code of your Python script. Run the script, and check the behaviour of the script. Add a function to produce a log file to record what has been done by your data reduction pipeline software. Make sure to print error messages once something wrong happens.