Advanced Astronomical Observations 2022
Session 14: Sky Background Brightness

Kinoshita Daisuke

06 May 2022
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 2022 to June 2022.
  - 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 measure sky background brightness.

1 Downloading and extracting reduced data

1.1 Downloading reduced data

If you have not yet finished the data reduction of the data taken on 14/Feb/2021, then download reduced data. The size of the data file is $\sim$ 6 GB. It may take a while ($\sim$ 20–30 min) to download. If you have already finished the data reduction, you do not need to download the file.

```
% curl -k -o lot_20210214r.tar.xz \
? https://s3b.astro.ncu.edu.tw/advobs_202202/data/lot_20210214r.tar.xz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 6035M  100 6035M    0     0  4198k      0  0:24:32  0:24:32 --:--:-- 4602k
% ls -lF lot_20210214r.tar.xz
-rw-r--r--  1 daisuke  taiwan  6329190792 May  5 16:59 lot_20210214r.tar.xz
% file lot_20210214r.tar.xz
lot_20210214r.tar.xz: XZ compressed data, checksum CRC64
```

1.2 Extracting reduced data

After the successful download, extract the data.

```
% tar xJvf lot_20210214r.tar.xz
x lot_20210214r/
x lot_20210214r/dark_00005000.fits
x lot_20210214r/dark_00010000.fits
x lot_20210214r/dark_00015000.fits
x lot_20210214r/dark_00045000.fits
x lot_20210214r/dark_00060000.fits
x lot_20210214r/dark_00180000.fits
x lot_20210214r/flat_gp_Astrodon_2019.fits
x lot_20210214r/flat_ip_Astrodon_2019.fits
x lot_20210214r/flat_rp_Astrodon_2019.fits
x lot_20210214r/lot_20210214_0085_df.fits
x lot_20210214r/lot_20210214_0086_df.fits
x lot_20210214r/lot_20210214_0087_df.fits
x lot_20210214r/lot_20210214_0088_df.fits
x lot_20210214r/lot_20210214_0089_df.fits
x lot_20210214r/lot_20210214_0093_df.fits
x lot_20210214r/lot_20210214_0094_df.fits
x lot_20210214r/lot_20210214_0095_df.fits
x lot_20210214r/lot_20210214_0096_df.fits
x lot_20210214r/lot_20210214_0097_df.fits
x lot_20210214r/lot_20210214_0098_df.fits
x lot_20210214r/lot_20210214_0099_df.fits
x lot_20210214r/lot_20210214_0100_df.fits

.....
```

2 Selecting and copying FITS files to process

Now, you have reduced data. Search for data suitable for sky background estimate and copy those data.

2.1 Selecting FITS files to process

We measure the sky background brightness. The data should be (1) an image of a photometric standard star field, and (2) an image of relatively longer exposure. Search for longer exposure data of a photometric standard star field from the data taken on 14 February 2021.

Make a Python script to select FITS files satisfying given criteria.

Python Code 1: advobs202202_s14_01.py

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

#
# Time-stamp: <2022/05/05 11:33:31 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

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

```
# constructing parser object
desc   = "selecting FITS files"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-e1', '--exptime-min', type=float, default=0.0, \
                     help='minimum exposure time in sec')
parser.add_argument ('-e2', '--exptime-max', type=float, default=3600.0, \
                     help='maximum exposure time in sec')
parser.add_argument ('-a', '--keyword-airmass', default='AIRMASS', \
                     help='FITS keyword for airmass (default: AIRMASS)')
parser.add_argument ('-d', '--keyword-datatype', default='IMAGETYP', \
                     help='FITS keyword for data type (default: IMAGETYP)')
parser.add_argument ('-e', '--keyword-exptime', default='EXPTIME', \
                     help='FITS keyword for exposure time (default: EXPTIME)')
parser.add_argument ('-f', '--keyword-filter', default='FILTER', \
                     help='FITS keyword for filter name (default: FILTER)')
parser.add_argument ('-o', '--keyword-object', default='OBJECT', \
                     help='FITS keyword for object name (default: OBJECT)')
parser.add_argument ('-t', '--keyword-timeobs', default='TIME-OBS', \
                     help='FITS keyword for TIME-OBS (default: TIME-OBS)')
parser.add_argument ('files', nargs='+', help='FITS files')

# parsing arguments
args = parser.parse_args ()

# input parameters
exptime_min      = args.exptime_min
exptime_max      = args.exptime_max
keyword_airmass  = args.keyword_airmass
keyword_datatype = args.keyword_datatype
keyword_exptime  = args.keyword_exptime
keyword_filter   = args.keyword_filter
keyword_object   = args.keyword_object
keyword_timeobs  = args.keyword_timeobs
files_fits       = args.files

# making an empty dictionary for storing search results
dic_fits = {}

# processing each FITS file
for file_fits in files_fits:
    # making pathlib object
    path_fits = pathlib.Path (file_fits)

    # check of file name
    if not (path_fits.suffix == '.fits'):
        # printing message
        print ("The file \"%s\" is not a FITS file!" % file_fits)
        print ("File must be a FITS file. Skipping.")
        # skipping
        continue

    # opening FITS file
    with astropy.io.fits.open (file_fits) as hdu_list:
        # reading header
        header = hdu_list[0].header
```

```
    # if data type is not 'LIGHT', then skip
    if not (header[keyword_datatype] == 'LIGHT'):
        continue

    # if exposure time does not match with specified exposure time criteria,
    # then skip
    if not ( (header[keyword_exptime] >= exptime_min) \
             and (header[keyword_exptime] <= exptime_max) ):
        continue

    # adding information to the dictionary
    dic_fits[file_fits] = {}
    dic_fits[file_fits][keyword_airmass]  = header[keyword_airmass]
    dic_fits[file_fits][keyword_datatype] = header[keyword_datatype]
    dic_fits[file_fits][keyword_exptime]  = header[keyword_exptime]
    dic_fits[file_fits][keyword_filter]   = header[keyword_filter]
    dic_fits[file_fits][keyword_object]   = header[keyword_object]
    dic_fits[file_fits][keyword_timeobs]  = header[keyword_timeobs]

# printing results
for file_fits in dic_fits.keys ():
    print (file_fits)
    print ("  %-8s = %-24s    %-8s = %-24s" \
           % (keyword_datatype, dic_fits[file_fits][keyword_datatype], \
              keyword_object, dic_fits[file_fits][keyword_object]) )
    print ("  %-8s = %-24s    %-8s = %f sec" \
           % (keyword_filter, dic_fits[file_fits][keyword_filter], \
              keyword_exptime, dic_fits[file_fits][keyword_exptime]) )
    print ("  %-8s = %-24s    %-8s = %f" \
           % (keyword_timeobs, dic_fits[file_fits][keyword_timeobs], \
              keyword_airmass, dic_fits[file_fits][keyword_airmass]) )
```

Execute the script, and search for FITS files that we look for.

```
% chmod a+x advobs_202202_s14_01.py
% ./advobs_202202_s14_01.py -h
usage: advobs_202202_s14_01.py [-h] [-e1 EXPTIME_MIN] [-e2 EXPTIME_MAX]
                               [-a KEYWORD_AIRMASS] [-d KEYWORD_DATATYPE]
                               [-e KEYWORD_EXPTIME] [-f KEYWORD_FILTER]
                               [-o KEYWORD_OBJECT] [-t KEYWORD_TIMEOBS]
                               files [files ...]

selecting FITS files

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -e1 EXPTIME_MIN, --exptime-min EXPTIME_MIN
                        minimum exposure time in sec
  -e2 EXPTIME_MAX, --exptime-max EXPTIME_MAX
                        maximum exposure time in sec
  -a KEYWORD_AIRMASS, --keyword-airmass KEYWORD_AIRMASS
                        FITS keyword for airmass (default: AIRMASS)
  -d KEYWORD_DATATYPE, --keyword-datatype KEYWORD_DATATYPE
                        FITS keyword for data type (default: IMAGETYP)
  -e KEYWORD_EXPTIME, --keyword-exptime KEYWORD_EXPTIME
```

```
                        FITS keyword for exposure time (default: EXPTIME)
  -f KEYWORD_FILTER, --keyword-filter KEYWORD_FILTER
                        FITS keyword for filter name (default: FILTER)
  -o KEYWORD_OBJECT, --keyword-object KEYWORD_OBJECT
                        FITS keyword for object name (default: OBJECT)
  -t KEYWORD_TIMEOBS, --keyword-timeobs KEYWORD_TIMEOBS
                        FITS keyword for TIME-OBS (default: TIME-OBS)

% ./advobs_202202_s14_01.py -e1 120 -e2 300 lot_20210214r/*.fits
lot_20210214r/lot_20210214_0185_df.fits
  IMAGETYP = LIGHT                          OBJECT   = PG1047
  FILTER   = gp_Astrodon_2019               EXPTIME  = 180.000000 sec
  TIME-OBS = 17:58:41                       AIRMASS  = 1.122154
lot_20210214r/lot_20210214_0186_df.fits
  IMAGETYP = LIGHT                          OBJECT   = PG1047
  FILTER   = gp_Astrodon_2019               EXPTIME  = 180.000000 sec
  TIME-OBS = 18:01:59                       AIRMASS  = 1.126160
lot_20210214r/lot_20210214_0187_df.fits
  IMAGETYP = LIGHT                          OBJECT   = PG1047
  FILTER   = rp_Astrodon_2019               EXPTIME  = 180.000000 sec
  TIME-OBS = 18:05:25                       AIRMASS  = 1.130614
lot_20210214r/lot_20210214_0188_df.fits
  IMAGETYP = LIGHT                          OBJECT   = PG1047
  FILTER   = rp_Astrodon_2019               EXPTIME  = 180.000000 sec
  TIME-OBS = 18:08:41                       AIRMASS  = 1.135143
lot_20210214r/lot_20210214_0189_df.fits
  IMAGETYP = LIGHT                          OBJECT   = PG1047
  FILTER   = ip_Astrodon_2019               EXPTIME  = 180.000000 sec
  TIME-OBS = 18:12:06                       AIRMASS  = 1.140164
lot_20210214r/lot_20210214_0190_df.fits
  IMAGETYP = LIGHT                          OBJECT   = PG1047
  FILTER   = ip_Astrodon_2019               EXPTIME  = 180.000000 sec
  TIME-OBS = 18:15:23                       AIRMASS  = 1.145258
```

We have found six FITS files. Those are data of the standard star field PG1047+003.

2.2 Copying FITS files to process

Make a Python script to copy FITS files that we have selected.

Python Code 2: advobs202202_s14_02.py

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

#
# Time-stamp: <2022/05/05 11:38:06 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing shutil module
import shutil
```

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

# constructing parser object
desc   = "copying FITS files"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-n', '--name', default='', help='target object name')
parser.add_argument ('-e1', '--exptime-min', type=float, default=0.0, \
                     help='minimum exposure time in sec')
parser.add_argument ('-e2', '--exptime-max', type=float, default=3600.0, \
                     help='maximum exposure time in sec')
parser.add_argument ('-a', '--keyword-airmass', default='AIRMASS', \
                     help='FITS keyword for airmass (default: AIRMASS)')
parser.add_argument ('-d', '--keyword-datatype', default='IMAGETYP', \
                     help='FITS keyword for data type (default: IMAGETYP)')
parser.add_argument ('-e', '--keyword-exptime', default='EXPTIME', \
                     help='FITS keyword for exposure time (default: EXPTIME)')
parser.add_argument ('-f', '--keyword-filter', default='FILTER', \
                     help='FITS keyword for filter name (default: FILTER)')
parser.add_argument ('-o', '--keyword-object', default='OBJECT', \
                     help='FITS keyword for object name (default: OBJECT)')
parser.add_argument ('-t', '--keyword-timeobs', default='TIME-OBS', \
                     help='FITS keyword for TIME-OBS (default: TIME-OBS)')
parser.add_argument ('files', nargs='+', help='FITS files')

# parsing arguments
args = parser.parse_args ()

# input parameters
target_name      = args.name
exptime_min      = args.exptime_min
exptime_max      = args.exptime_max
keyword_airmass  = args.keyword_airmass
keyword_datatype = args.keyword_datatype
keyword_exptime  = args.keyword_exptime
keyword_filter   = args.keyword_filter
keyword_object   = args.keyword_object
keyword_timeobs  = args.keyword_timeobs
files_fits       = args.files

# check of target object name
if (target_name == ''):
    print ("Target object name must be specified.")
    sys.exit ()

# making an empty dictionary for storing search results
dic_fits = {}

# processing each FITS file
for file_fits in files_fits:
    # making pathlib object
    path_fits = pathlib.Path (file_fits)

    # check of file name
    if not (path_fits.suffix == '.fits'):
        # printing message
```

```
        print ("The file \"%s\" is not a FITS file!" % file_fits)
        print ("File must be a FITS file. Skipping.")
        # skipping
        continue

    # opening FITS file
    with astropy.io.fits.open (file_fits) as hdu_list:
        # reading header
        header = hdu_list[0].header

    # if data type is not 'LIGHT', then skip
    if not (header[keyword_datatype] == 'LIGHT'):
        continue

    # if target object name does not match with specified target object name,
    # then skip
    if not (header[keyword_object] == target_name):
        continue

    # if exposure time does not match with specified exposure time criteria,
    # then skip
    if not ( (header[keyword_exptime] >= exptime_min) \
             and (header[keyword_exptime] <= exptime_max) ):
        continue

    # adding information to the dictionary
    dic_fits[file_fits] = {}
    dic_fits[file_fits][keyword_airmass]  = header[keyword_airmass]
    dic_fits[file_fits][keyword_datatype] = header[keyword_datatype]
    dic_fits[file_fits][keyword_exptime]  = header[keyword_exptime]
    dic_fits[file_fits][keyword_filter]   = header[keyword_filter]
    dic_fits[file_fits][keyword_object]   = header[keyword_object]
    dic_fits[file_fits][keyword_timeobs]  = header[keyword_timeobs]

# copying files
for file_fits in dic_fits.keys ():
    # making pathlib object
    path_fits = pathlib.Path (file_fits)
    # if the file exists, then copy the file to currently working directory
    if (path_fits.exists () ):
        # printing status
        print ("Processing the file \"%s\"..." % path_fits.name)
        print ("  now copying the file \"%s\"..." % path_fits.name)
        # copying file
        shutil.copy2 (path_fits, '.')
        print ("  finished copying the file \"%s\"!" % path_fits.name)
```

Run the script, and copy files to the currently working directory.

```
% chmod a+x advobs_202202_s14_02.py
% ./advobs_202202_s14_02.py -h
usage: advobs_202202_s14_02.py [-h] [-n NAME] [-e1 EXPTIME_MIN]
                               [-e2 EXPTIME_MAX] [-a KEYWORD_AIRMASS]
                               [-d KEYWORD_DATATYPE] [-e KEYWORD_EXPTIME]
                               [-f KEYWORD_FILTER] [-o KEYWORD_OBJECT]
                               [-t KEYWORD_TIMEOBS]
                               files [files ...]
```

```
copying FITS files

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -n NAME, --name NAME  target object name
  -e1 EXPTIME_MIN, --exptime-min EXPTIME_MIN
                        minimum exposure time in sec
  -e2 EXPTIME_MAX, --exptime-max EXPTIME_MAX
                        maximum exposure time in sec
  -a KEYWORD_AIRMASS, --keyword-airmass KEYWORD_AIRMASS
                        FITS keyword for airmass (default: AIRMASS)
  -d KEYWORD_DATATYPE, --keyword-datatype KEYWORD_DATATYPE
                        FITS keyword for data type (default: IMAGETYP)
  -e KEYWORD_EXPTIME, --keyword-exptime KEYWORD_EXPTIME
                        FITS keyword for exposure time (default: EXPTIME)
  -f KEYWORD_FILTER, --keyword-filter KEYWORD_FILTER
                        FITS keyword for filter name (default: FILTER)
  -o KEYWORD_OBJECT, --keyword-object KEYWORD_OBJECT
                        FITS keyword for object name (default: OBJECT)
  -t KEYWORD_TIMEOBS, --keyword-timeobs KEYWORD_TIMEOBS
                        FITS keyword for TIME-OBS (default: TIME-OBS)

% ./advobs_202202_s14_02.py -n PG1047 -e1 120 -e2 300 lot_20210214r/*.fits
Processing the file "lot_20210214_0185_df.fits"...
  now copying the file "lot_20210214_0185_df.fits"...
  finished copying the file "lot_20210214_0185_df.fits"!
Processing the file "lot_20210214_0186_df.fits"...
  now copying the file "lot_20210214_0186_df.fits"...
  finished copying the file "lot_20210214_0186_df.fits"!
Processing the file "lot_20210214_0187_df.fits"...
  now copying the file "lot_20210214_0187_df.fits"...
  finished copying the file "lot_20210214_0187_df.fits"!
Processing the file "lot_20210214_0188_df.fits"...
  now copying the file "lot_20210214_0188_df.fits"...
  finished copying the file "lot_20210214_0188_df.fits"!
Processing the file "lot_20210214_0189_df.fits"...
  now copying the file "lot_20210214_0189_df.fits"...
  finished copying the file "lot_20210214_0189_df.fits"!
Processing the file "lot_20210214_0190_df.fits"...
  now copying the file "lot_20210214_0190_df.fits"...
  finished copying the file "lot_20210214_0190_df.fits"!
% ls *.fits
lot_20210214_0185_df.fits               lot_20210214_0188_df.fits
lot_20210214_0186_df.fits               lot_20210214_0189_df.fits
lot_20210214_0187_df.fits               lot_20210214_0190_df.fits
```

Six FITS files of PG1047+003 field has been copied.

2.3 Visual inspection of the image

We first process the file `lot_20210214_0185_df.fits`. Use Ginga to show the image of tt lot_20210214_0185_df.fits. (Fig. 1)

```
% ginga lot_20210214_0185_df.fits &
```

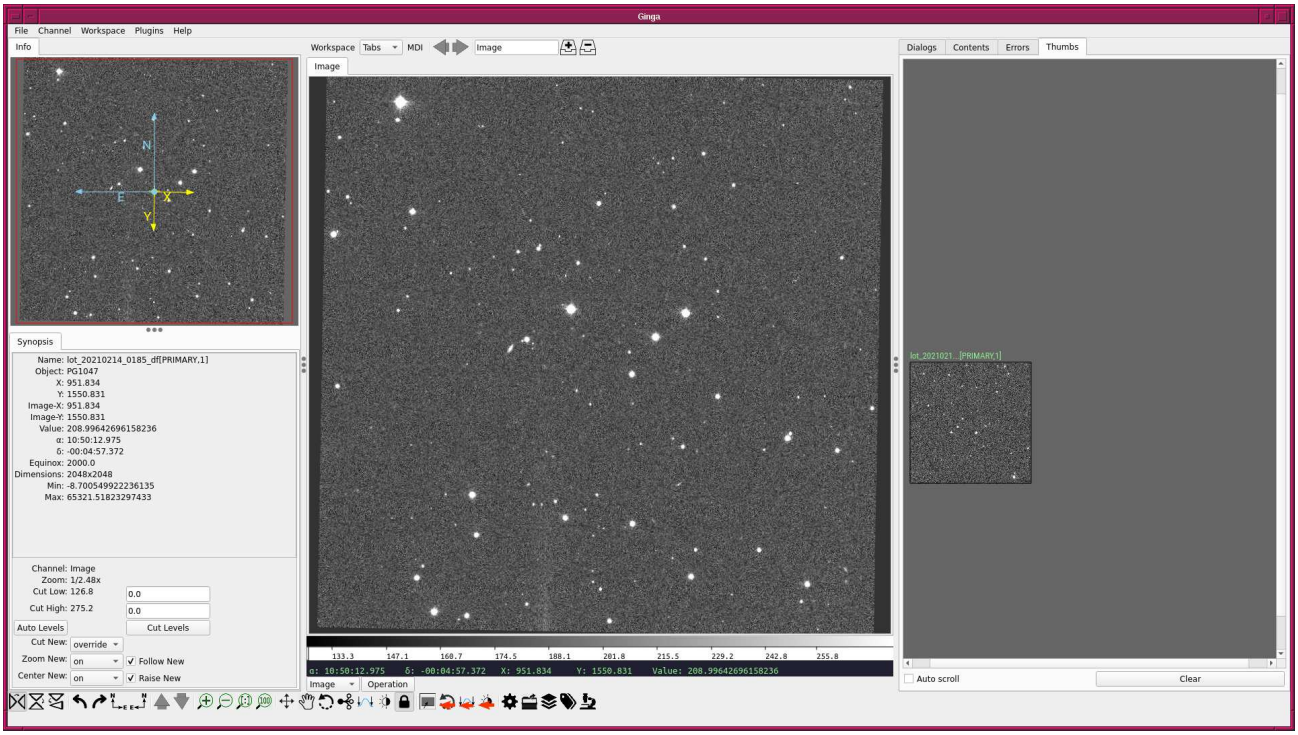

Figure 1: The image of `lot_20210214_0185_df.fits` displayed using Ginga.

3 Photometric standard stars

3.1 Downloading photometric standard star catalogue

Download the photometric standard star catalogue.

```
% curl -k -o stds.tar.gz \
? https://www-star.fnal.gov/NorthEqExtension_ugriz/Data/usno40stds.clean.v3.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 65774  100 65774    0     0  35590      0  0:00:01  0:00:01 --:--:-- 35591
% ls -lF stds.tar.gz
-rw-r--r--  1 daisuke  taiwan  65774 May  5 11:49 stds.tar.gz
```

3.2 Extracting photometric standard star catalogue

Extract the photometric standard star catalogue.

```
% mkdir stds
% ls
advobs_202202_s14_01.py*                lot_20210214_0188_df.fits
advobs_202202_s14_01.py~                lot_20210214_0189_df.fits
advobs_202202_s14_02.py*                lot_20210214_0190_df.fits
advobs_202202_s14_02.py~                lot_20210214r@
lot_20210214_0185_df.fits               stds/
lot_20210214_0186_df.fits               stds.tar.gz
lot_20210214_0187_df.fits
% cd stds
% ls
% tar xzvf ../stds.tar.gz
x 100_a.txt.clean.v3
```

```
x 100_b.txt.clean.v3
x 101_a.txt.clean.v3
x 101_c.txt.clean.v3
x 104_a.txt.clean.v3
x 105_a.txt.clean.v3
x 107_a.txt.clean.v3
x 107_b.txt.clean.v3
x 108_a.txt.clean.v3
x 108_b.txt.clean.v3

.....

x Ross374.txt.clean.v3
x Ross49.txt.clean.v3
x Ross530.txt.clean.v3
x Ross683.txt.clean.v3
x Ross711.txt.clean.v3
x Ross838.txt.clean.v3
x Ru149.txt.clean.v3
x Wolf1346.txt.clean.v3
x Wolf1447.txt.clean.v3
x Wolf365.txt.clean.v3
% ls
100_a.txt.clean.v3          97_a.txt.clean.v3          GCRV5951.txt.clean.v3
100_b.txt.clean.v3          97_b.txt.clean.v3          GCRV7017.txt.clean.v3
101_a.txt.clean.v3          97_c.txt.clean.v3          GCRV7951.txt.clean.v3
101_c.txt.clean.v3          97_d.txt.clean.v3          GCRV8758.txt.clean.v3
104_a.txt.clean.v3          98_a.txt.clean.v3          GCRV9438.txt.clean.v3

.....

95_b.txt.clean.v3           G15-24.txt.clean.v3        Ru149.txt.clean.v3
95_f.txt.clean.v3           G163_50-51.txt.clean.v3    Wolf1346.txt.clean.v3
96_a.txt.clean.v3           G27-45.txt.clean.v3        Wolf1447.txt.clean.v3
96_b.txt.clean.v3           G3-33.txt.clean.v3         Wolf365.txt.clean.v3
96_c.txt.clean.v3           GCRV5757.txt.clean.v3
% cd ..
```

3.3 Showing coordinates of photometric standards in PG 1047+003 field

Make a Python script to show the coordinates of photometric standard stars in the field PG 1047+003.

Python Code 3: advobs202202_s14_03.py

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

#
# Time-stamp: <2022/05/05 11:57:16 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib
```

```
# importing astropy module
import astropy.coordinates

# constructing parser object
desc   = "showing coordinates of photometric standard stars"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('files', nargs='+', help='photometric standard star files')

# parsing arguments
args = parser.parse_args ()

# input parameters
list_files = args.files

# printing header
print ("# ID, RA (deg), Dec (deg), RA (hex), Dec (hex), g' mag, r' mag, i' mag")

# processing each catalogue file
for file_cat in list_files:
    # making pathlib object
    path_cat = pathlib.Path (file_cat)

    # existence check of catalogue file
    if not (path_cat.exists ()):
        # printing message
        print ("ERROR: the file '%s' does not exist!" % file_cat)
        # exit
        sys.exit ()

    # opening the file
    with open (file_cat, 'r') as fh:
        # reading the file line-by-line
        for line in fh:
            # if the line starts with '#', then skip
            if (line[0] == '#'):
                continue
            # splitting the data
            records = line.split ()
            # star ID
            star_id = int (records[0])
            # RA
            ra_deg  = float (records[1])
            # Dec
            dec_deg = float (records[2])
            # g'-band magnitude
            mag_g   = float (records[6])
            # r'-band magnitude
            mag_r   = float (records[9])
            # i'-band magnitude
            mag_i   = float (records[12])
            # coordinates
            coord = astropy.coordinates.SkyCoord (ra_deg, dec_deg, unit='deg')
            # conversion into hmsdms format
            radec_str = coord.to_string ('hmsdms')
            (ra_str, dec_str) = radec_str.split ()
            # printing coordinate
            print ("%03d %9.5f %+8.4f %-16s %-16s %6.3f %6.3f %6.3f" \
```

```
                % (star_id, ra_deg, dec_deg, ra_str, dec_str, \
                   mag_g, mag_r, mag_i) )
```

Execute the script, and show coordinates of stars in the field of PG 1047+003.

```
% chmod a+x advobs_202202_s14_03.py
% ./advobs_202202_s14_03.py -h
usage: advobs_202202_s14_03.py [-h] files [files ...]

showing coordinates of photometric standard stars

positional arguments:
  files       photometric standard star files

optional arguments:
  -h, --help  show this help message and exit

% ./advobs_202202_s14_03.py stds/PG1047+003A.txt.clean.v3
# ID, RA (deg), Dec (deg), RA (hex), Dec (hex), g' mag, r' mag, i' mag
004 162.62371  +0.0725 10h50m29.6904s   +00d04m21s       12.127 11.709 11.587
005 162.55708  -0.0090 10h50m13.6992s   -00d00m32.4s     12.826 12.373 12.217
006 162.52363  -0.0198 10h50m05.6712s   -00d01m11.28s    13.819 13.304 13.092
007 162.51184  -0.0102 10h50m02.8416s   -00d00m36.72s    13.240 13.718 14.087
008 162.59599  -0.0818 10h50m23.0376s   -00d04m54.48s    14.511 13.803 13.579
010 162.47199  -0.0596 10h49m53.2776s   -00d03m34.56s    14.797 14.308 14.139
011 162.47082  -0.0579 10h49m52.9968s   -00d03m28.44s    14.814 14.318 14.143
013 162.53305  -0.0346 10h50m07.932s    -00d02m04.56s    15.061 14.553 14.409
014 162.53285  -0.0932 10h50m07.884s    -00d05m35.52s    15.157 14.616 14.440
017 162.57442  -0.0208 10h50m17.8608s   -00d01m14.88s    15.222 14.838 14.707
020 162.61912  +0.0295 10h50m28.5888s   +00d01m46.2s     14.961 14.897 14.946
022 162.55930  -0.0909 10h50m14.232s    -00d05m27.24s    15.415 14.933 14.772
025 162.49926  -0.0433 10h49m59.8224s   -00d02m35.88s    15.861 15.322 15.123
```

13 stars are listed.

3.4 Marking photometric standard stars

Make a Python script to mark photometric standard stars using Matplotlib package.

Python Code 4: advobs202202_s14_04.py

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

#
# Time-stamp: <2022/05/05 12:39:26 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing numpy module
import numpy
```

```
# importing astropy module
import astropy.coordinates
import astropy.units
import astropy.wcs

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# constructing parser object
desc   = 'marking target objects'
parser = argparse.ArgumentParser (description=desc)

# colour maps
choices_cmap = ['viridis', 'plasma', 'inferno', 'magma', 'cividis', \
                'binary', 'gray', 'bone', 'pink', \
                'spring', 'summer', 'autumn', 'winter', \
                'cool', 'hot', 'copper', 'ocean', 'terrain', \
                'gnuplot', 'cubehelix', 'jet', 'turbo']

# adding argument
parser.add_argument ('-i', '--input', default='', \
                     help='input FITS file name')
parser.add_argument ('-o', '--output', default='', \
                     help='output image file name')
parser.add_argument ('-c', '--catalogue', default='', \
                     help='standard star catalogue file name')
parser.add_argument ('-r', '--radius', type=float, default=10.0, \
                     help='radius of aperture in pixel (default: 10)')
parser.add_argument ('-w', '--width', type=float, default=2.0, \
                     help='width of circle (default: 2)')
parser.add_argument ('-l', '--resolution', type=int, default=450, \
                     help='resolution of output image in DPI (default: 450)')
parser.add_argument ('-m', '--cmap', default='bone', choices=choices_cmap, \
                     help='choice of colour map (default: bone)')

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

# input parameters
file_cat    = args.catalogue
file_fits   = args.input
file_output = args.output
radius_pix  = args.radius
width       = args.width
resolution  = args.resolution
cmap        = args.cmap

# making pathlib objects
path_cat    = pathlib.Path (file_cat)
path_fits   = pathlib.Path (file_fits)
path_output = pathlib.Path (file_output)

# if output file name is not given, then stop the script
if (file_output == ''):
    # printing message
    print ("ERROR: output file name has to be given.")
    # exit
```

```
    sys.exit ()

# existence checks of files
if not (path_cat.exists ()):
    # printing message
    print ("ERROR: the file '%s' does not exist!" % file_cat)
    # exit
    sys.exit ()
if not (path_fits.exists ()):
    # printing message
    print ("ERROR: the file '%s' does not exist!" % file_fits)
    # exit
    sys.exit ()
if (path_output.exists ()):
    # printing message
    print ("ERROR: the file '%s' exists!" % file_output)
    # exit
    sys.exit ()

# check of FITS file name
if not (path_fits.suffix == '.fits'):
    # printing message
    print ("ERROR: The file \"%s\" is not a FITS file!" % file_fits)
    print ("ERROR: Check the file name.")
    # exit
    sys.exit ()

# check of output image file
if not ( (path_output.suffix == '.eps') or (path_output.suffix == '.pdf') \
         or (path_output.suffix == '.png') or (path_output.suffix == '.ps') ):
    # printing message
    print ("Output image file must be either EPS, PDF, PNG, or PS.")
    print ("Given output image file name = %s" % file_output)
    # exit
    sys.exit ()

# check of catalogue file
if not ( (path_cat.suffixes[0] == '.txt') \
         and (path_cat.suffixes[1] == '.clean') \
         and (path_cat.suffixes[2] == '.v3') ):
    # printing message
    print ("ERROR: Catalogue file must be SDSS standard star catalogue.")
    print ("ERROR: Given catalogue file name = %s" % file_cat)
    # exit
    sys.exit ()

# making an empty dictionary to store data for standards
dic_stds = {}

# opening catalogue file
with open (file_cat, 'r') as fh:
    # reading the file line-by-line
    for line in fh:
        # if the line stars with '#', then skip
        if (line[0] == '#'):
            continue
        # splitting the data
        records = line.split ()
        # star ID
```

```
        star_id = int (records[0])
        # RA
        ra_deg  = float (records[1])
        # Dec
        dec_deg = float (records[2])
        # adding data to the dictionary
        if not (star_id in dic_stds):
            dic_stds[star_id] = {}
            dic_stds[star_id]['RA_deg'] = ra_deg
            dic_stds[star_id]['Dec_deg'] = dec_deg

# positions of standard stars
positions = []
for star in dic_stds.keys ():
    positions.append ( (dic_stds[star]['RA_deg'], dic_stds[star]['Dec_deg']) )

# making Astropy's coordinate object
coords = astropy.coordinates.SkyCoord (positions, unit='deg')

# opening FITS file
with astropy.io.fits.open (file_fits) as hdu_list:
    # reading header information
    header = hdu_list[0].header
    # WCS information
    wcs = astropy.wcs.WCS (header)
    # reading image data
    data   = hdu_list[0].data

# (x, y) coordinate of standard star on image
(x, y) = wcs.world_to_pixel (coords)

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax = fig.add_subplot (111, projection=wcs)

# axes
ax.set_xlabel ('RA')
ax.set_ylabel ('Dec')

# plotting image
norm \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (data) )
im = ax.imshow (data, origin='lower', cmap=cmap, norm=norm)
fig.colorbar (im)

# plotting each standard star location
for i in range ( len (x) ):
    # making a circle to indicate the location of standard star
    stdstars = matplotlib.patches.Circle (xy=(x[i], y[i]), \
                                          radius=radius_pix, \
                                          fill=False, color="red", \
                                          linewidth=width)
    # plotting location of standard star
    ax.add_patch (stdstars)

# invert Y-axis
ax.invert_yaxis ()
```

```
# saving file
fig.savefig (file_output, dpi=resolution)
```

Run the script, and make a PDF file.

```
% chmod a+x advobs_202202_s14_04.py
% ./advobs_202202_s14_04.py -h
usage: advobs_202202_s14_04.py [-h] [-i INPUT] [-o OUTPUT] [-c CATALOGUE]
                               [-r RADIUS] [-w WIDTH] [-l RESOLUTION]
                               [-m {viridis,plasma,inferno,magma,cividis,binary,
gray,bone,pink,spring,summer,autumn,winter,cool,hot,copper,ocean,terrain,gnuplot
,cubehelix,jet,turbo}]

marking target objects

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        input FITS file name
  -o OUTPUT, --output OUTPUT
                        output image file name
  -c CATALOGUE, --catalogue CATALOGUE
                        standard star catalogue file name
  -r RADIUS, --radius RADIUS
                        radius of aperture in pixel (default: 10)
  -w WIDTH, --width WIDTH
                        width of circle (default: 2)
  -l RESOLUTION, --resolution RESOLUTION
                        resolution of output image in DPI (default: 450)
  -m {viridis,plasma,inferno,magma,cividis,binary,gray,bone,pink,spring,summer,a
utumn,winter,cool,hot,copper,ocean,terrain,gnuplot,cubehelix,jet,turbo}, --cmap
{viridis,plasma,inferno,magma,cividis,binary,gray,bone,pink,spring,summer,autumn
,winter,cool,hot,copper,ocean,terrain,gnuplot,cubehelix,jet,turbo}
                        choice of colour map (default: bone)

% ./advobs_202202_s14_04.py -i lot_20210214_0185_df.fits \
? -c stds/PG1047+003A.txt.clean.v3 -r 50 -m viridis -o pg1047_0185.png
WARNING: FITSFixedWarning: RADECSYS= 'FK5 ' / Equatorial coordinate system
the RADECSYS keyword is deprecated, use RADESYSa. [astropy.wcs.wcs]
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to 59259.749086
 from DATE-OBS'. [astropy.wcs.wcs]
% ls -lF pg1047_0185.png
-rw-r--r--  1 daisuke  taiwan  8652880 May  5 12:47 pg1047_0185.png
```

Show the image file using the command `feh`. (Fig. 2)

```
% feh -dF pg1047_0185.png
```

4 g'-band sky background brightness

We first measure g'-band sky background brightness.

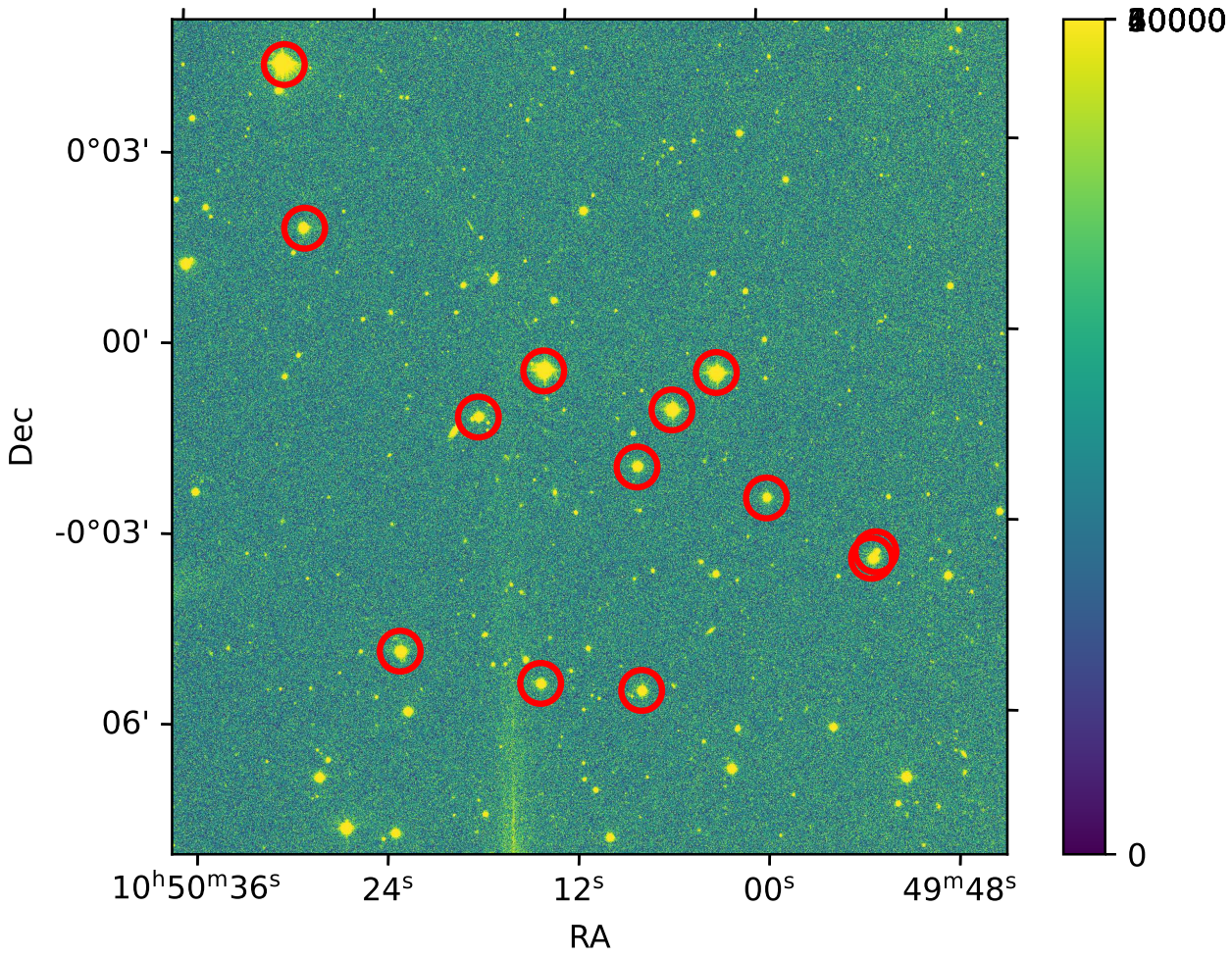

Figure 2: Photometric standard stars in the field of PG1047+003.

4.1 Selecting a standard star

Check peak counts of standard stars, and select a star for the measurement.
Use Ginga to display the image.

```
% ginga lot_20210214_0185_df.fits &
```

Click the menu 'Plugins', move your mouse cursor to 'Analysis', then choose 'Pick'. (Fig. 3) Move your mouse cursor to the position of one of photometric standard stars in PG1047+003 field, and click the left-button of your mouse. (Fig. 4) Show the radial plot by clicking the button "Radial". (Fig. 5) If the peak count is $\sim$ 65,000 ADU, the image of the star is saturated. Saturated stars cannot be used for the measurement. Find a photometric standard star with the peak count of $\sim$ 10,000 – 20,000 ADU.

For following analysis, we use the star #025 at $(RA, Dec) = (10^h49^m59.8^s, -00°02'36'')$. (Fig. 6)

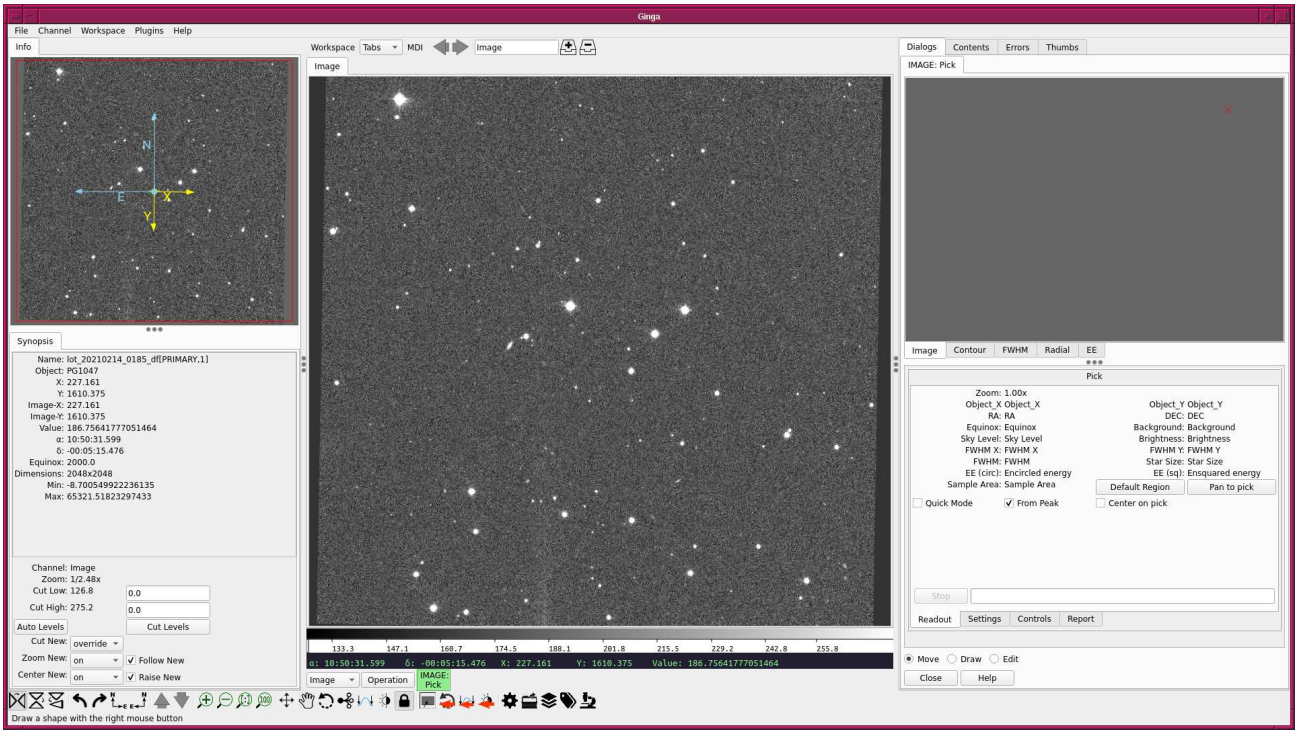

Figure 3: Selecting "Pick" from the menu "Analysis"

4.2 Aperture photometry of a standard star

Make a Python script to carry out aperture photometry of a selected photometric standard star.

Python Code 5: advobs202202_s14_05.py

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

#
# Time-stamp: <2022/05/05 13:05:34 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys
```

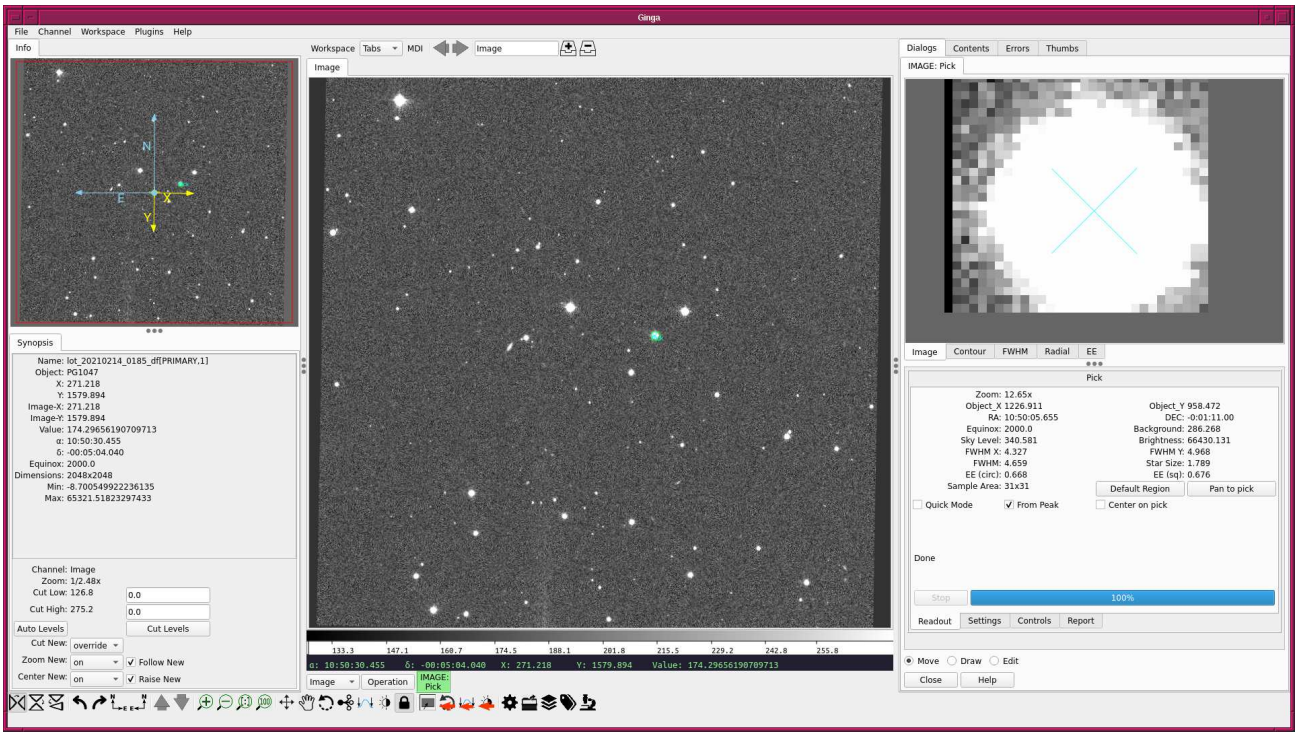

Figure 4: Trying “Pick” for one of standard stars.

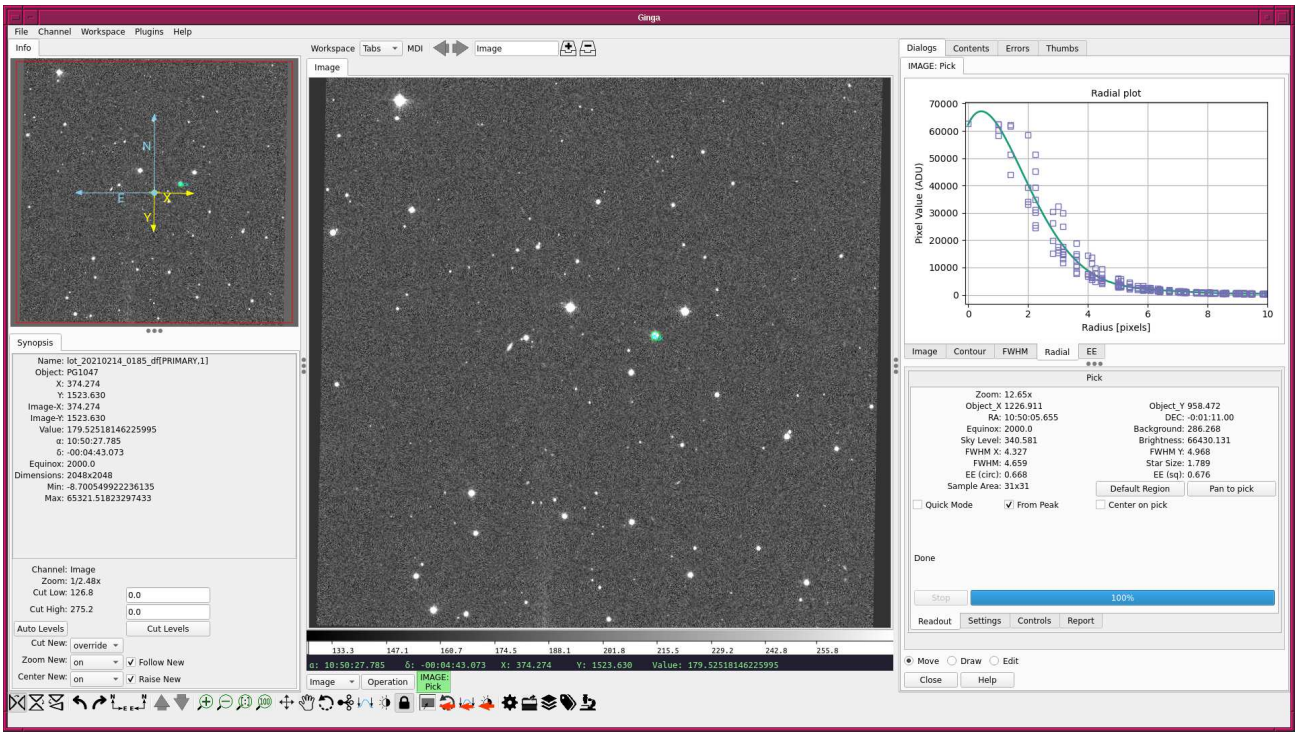

Figure 5: The radial profile of one of standard stars.

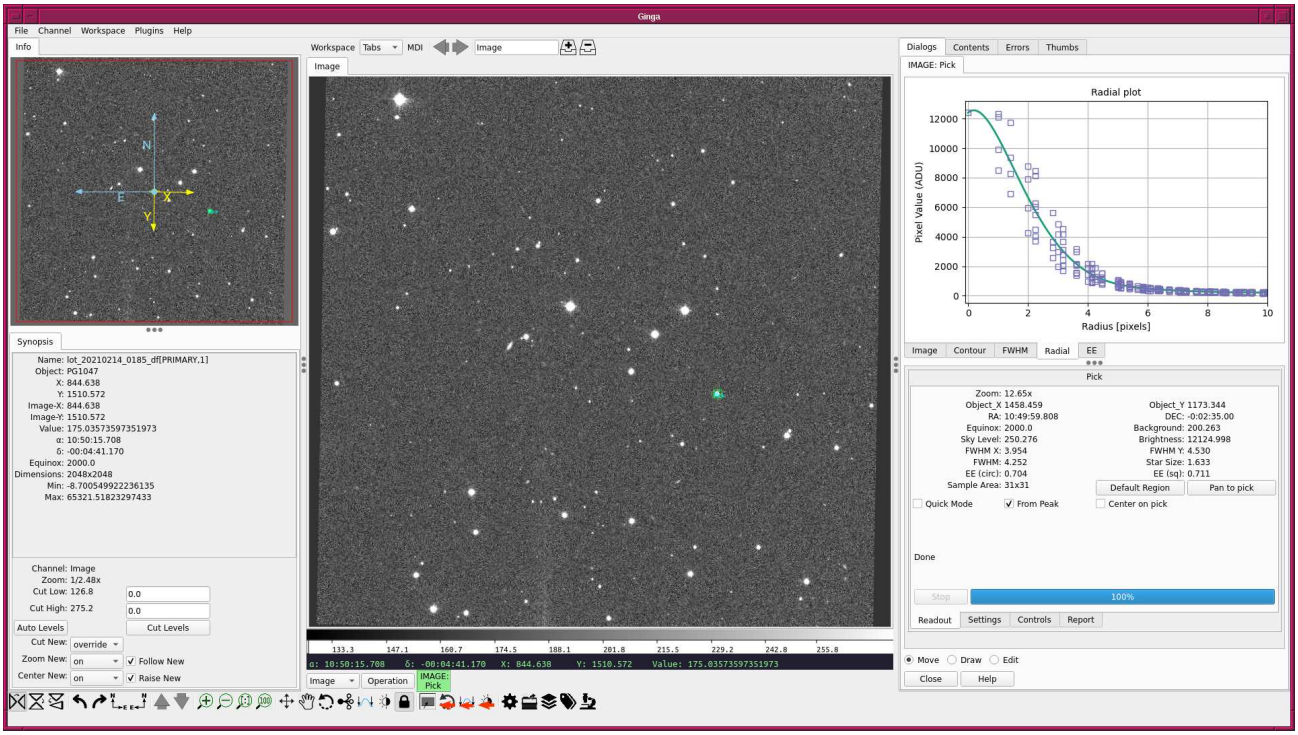

Figure 6: The radial profile of the standard stars ID 025.

```
# importing pathlib module
import pathlib

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.coordinates
import astropy.stats

# importing photutils module
import photutils.centroids
import photutils.aperture

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg
import matplotlib.patches

# constructing parser object
desc   = 'aperture photometry of a star at given RA and Dec'
parser = argparse.ArgumentParser (description=desc)

# centroid measurement technique
choices_centroid = ['com', '1dg', '2dg']

# PSF models (Gaussian and Moffat)
choices_psf = ['2dg', '2dm']
```

```
# colour maps
choices_cmap = ['viridis', 'plasma', 'inferno', 'magma', 'cividis', \
                'binary', 'gray', 'bone', 'pink', \
                'spring', 'summer', 'autumn', 'winter', \
                'cool', 'hot', 'copper', 'ocean', 'terrain', \
                'gnuplot', 'cubehelix', 'jet', 'turbo']

# adding argument
parser.add_argument ('-i', '--input', default='', \
                     help='input FITS file name')
parser.add_argument ('-o', '--output', default='', \
                     help='output data file name')
parser.add_argument ('-g', '--graphic', default='', \
                     help='output graphic file name')
parser.add_argument ('-c', '--centroid', choices=choices_centroid, \
                     default='2dg', \
                     help='centroid measurement algorithm (default: 2dg)')
parser.add_argument ('-p', '--psf', choices=choices_psf, default='2dg', \
                     help='PSF model [2dg=Gaussian, 2dm=Moffat] (default: 2dg)')
parser.add_argument ('-r', '--ra', type=float, default=-999.999, \
                     help='RA in degree')
parser.add_argument ('-d', '--dec', type=float, default=-999.999, \
                     help='Dec in degree')
parser.add_argument ('-a', '--aperture', type=float, default=2.0, \
                     help='aperture radius in FWHM (default: 2.0)')
parser.add_argument ('-w', '--halfwidth', type=int, default=20, \
                     help='half-width for centroid measurement (default: 20)')
parser.add_argument ('-s1', '--skyannulus1', type=float, default=4.0, \
                     help='inner sky annulus radius in FWHM (default: 4)')
parser.add_argument ('-s2', '--skyannulus2', type=float, default=7.0, \
                     help='outer sky annulus radius in FWHM (default: 7)')
parser.add_argument ('-t', '--threshold', type=float, default=4.0, \
                     help='threshold for sigma-clipping in sigma (default: 4)')
parser.add_argument ('-n', '--maxiters', type=int, default=100, \
                     help='maximum number of iterations (default: 100)')
parser.add_argument ('-e', '--keyword-exptime', default='EXPTIME', \
                     help='FITS keyword for exposure time (default: EXPTIME)')
parser.add_argument ('-f', '--keyword-filter', default='FILTER', \
                     help='FITS keyword for filter name (default: FILTER)')
parser.add_argument ('-m', '--keyword-airmass', default='AIRMASS', \
                     help='FITS keyword for airmass (default: AIRMASS)')
parser.add_argument ('-z', '--cmap', default='bone', choices=choices_cmap, \
                     help='choice of colour map (default: bone)')
parser.add_argument ('-l', '--resolution', type=int, default=450, \
                     help='resolution in DPI (default: 450)')

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

# input parameters
file_fits             = args.input
file_output           = args.output
file_graphic          = args.graphic
centroid              = args.centroid
psf_model             = args.psf
target_ra_deg         = args.ra
target_dec_deg        = args.dec
aperture_radius_fwhm  = args.aperture
```

```
halfwidth              = args.halfwidth
skyannulus_inner_fwhm = args.skyannulus1
skyannulus_outer_fwhm = args.skyannulus2
threshold              = args.threshold
maxiters               = args.maxiters
keyword_exptime        = args.keyword_exptime
keyword_filter         = args.keyword_filter
keyword_airmass        = args.keyword_airmass
cmap                   = args.cmap
resolution             = args.resolution

# check of RA and Dec
if ( (target_ra_deg < 0.0) or (target_ra_deg > 360.0) \
     or (target_dec_deg < -90.0) or (target_dec_deg > 90.0) ):
    # printing message
    print ("Something is wrong with RA or Dec!")
    print ("Check RA and Dec you specify.")
    print ("RA  = %f deg" % target_ra_deg)
    print ("Dec = %f deg" % target_dec_deg)
    # exit
    sys.exit ()

# making pathlib objects
path_fits    = pathlib.Path (file_fits)
path_output  = pathlib.Path (file_output)
path_graphic = pathlib.Path (file_graphic)

# existence checks
if not (path_fits.exists ()):
    # printing message
    print ("The file \"%s\" does not exist!")
    # exit
    sys.exit ()
if (path_output.exists ()):
    # printing message
    print ("The file \"%s\" exists!" % file_output)
    # exit
    sys.exit ()
if (path_graphic.exists ()):
    # printing message
    print ("The file \"%s\" exists!" % file_graphic)
    # exit
    sys.exit ()

# check of FITS file name
if not (path_fits.suffix == '.fits'):
    # printing message
    print ("The file \"%s\" is not a FITS file!" % file_fits)
    print ("Check the file name.")
    # exit
    sys.exit ()

# check of output file
if (file_output == ''):
    # printing message
    print ("Output file name must be given.")
    # exit
    sys.exit ()
```

```
# check of output graphic file
if not ( (path_graphic.suffix == '.eps') or (path_graphic.suffix == '.pdf') \
         or (path_graphic.suffix == '.png') or (path_graphic.suffix == '.ps') ):
    # printing message
    print ("Output graphic file must be either EPS, PDF, PNG, or PS.")
    print ("Given graphic file name = %s" % file_graphic)
    # exit
    sys.exit ()

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

# opening FITS file
with astropy.io.fits.open (file_fits) as hdu_list:
    # reading header information
    header = hdu_list[0].header
    # WCS information
    wcs = astropy.wcs.WCS (header)
    # reading image data
    data   = hdu_list[0].data

# extraction of information from FITS header
exptime      = header[keyword_exptime]
filter_name  = header[keyword_filter]
airmass      = header[keyword_airmass]

# sky coordinate
coord_sky = astropy.coordinates.SkyCoord (target_ra_deg, target_dec_deg, \
                                          unit='deg')

# conversion from sky coordinate into pixel coordinate
coord_pix = wcs.world_to_pixel (coord_sky)

# initial guess of target position
init_x = coord_pix[0]
init_y = coord_pix[1]

# region of sub-frame for centroid measurement
subframe_xmin = int (init_x - halfwidth)
subframe_xmax = int (init_x + halfwidth + 1)
subframe_ymin = int (init_y - halfwidth)
subframe_ymax = int (init_y + halfwidth + 1)

# extracting sub-frame for centroid measurement
subframe = data[subframe_ymin:subframe_ymax, subframe_xmin:subframe_xmax]

# sky subtraction
subframe_skysub = subframe - numpy.median (subframe)

# centroid measurement
if (centroid == 'com'):
#    (x_centre, y_centre) = photutils.centroids.centroid_com (subframe_skysub)
    (x_centre, y_centre) = photutils.centroids.centroid_com (subframe)
elif (centroid == '1dg'):
#    (x_centre, y_centre) = photutils.centroids.centroid_1dg (subframe_skysub)
    (x_centre, y_centre) = photutils.centroids.centroid_1dg (subframe)
elif (centroid == '2dg'):
#    (x_centre, y_centre) = photutils.centroids.centroid_2dg (subframe_skysub)
    (x_centre, y_centre) = photutils.centroids.centroid_2dg (subframe)
```

```
# PSF fitting
subframe_y, subframe_x = numpy.indices (subframe_skysub.shape)
if (psf_model == '2dg'):
    psf_init = astropy.modeling.models.Gaussian2D (x_mean=x_centre, \
                                                   y_mean=y_centre)
elif (psf_model == '2dm'):
    psf_init = astropy.modeling.models.Moffat2D (x_0=x_centre, y_0=y_centre, \
                                                 amplitude=1.0, \
                                                 alpha=1.0, gamma=1.0)
fit = astropy.modeling.fitting.LevMarLSQFitter ()
psf_fitted = fit (psf_init, subframe_x, subframe_y, subframe_skysub, \
                  maxiter=maxiters)

# fitted PSF parameters
amplitude = psf_fitted.amplitude.value
if (psf_model == '2dg'):
    x_centre_sub = psf_fitted.x_mean.value
    y_centre_sub = psf_fitted.y_mean.value
    x_centre_psf = psf_fitted.x_mean.value + subframe_xmin
    y_centre_psf = psf_fitted.y_mean.value + subframe_ymin
    x_fwhm       = psf_fitted.x_fwhm
    y_fwhm       = psf_fitted.y_fwhm
    fwhm         = (x_fwhm + y_fwhm) / 2.0
    theta        = psf_fitted.theta.value
if (psf_model == '2dm'):
    x_centre_sub = psf_fitted.x_0.value
    y_centre_sub = psf_fitted.y_0.value
    x_centre_psf = psf_fitted.x_0.value + subframe_xmin
    y_centre_psf = psf_fitted.y_0.value + subframe_ymin
    alpha        = psf_fitted.alpha.value
    gamma        = psf_fitted.gamma.value
    fwhm         = psf_fitted.fwhm

# position of centre of star in pixel coordinate
position_pix = (x_centre_psf, y_centre_psf)

# aperture radius in pixel
aperture_radius_pix  = fwhm * aperture_radius_fwhm
skyannulus_inner_pix = fwhm * skyannulus_inner_fwhm
skyannulus_outer_pix = fwhm * skyannulus_outer_fwhm

# making aperture
apphot_aperture \
    = photutils.aperture.CircularAperture (position_pix, r=aperture_radius_pix)
apphot_annulus \
    = photutils.aperture.CircularAnnulus (position_pix, \
                                          r_in=skyannulus_inner_pix, \
                                          r_out=skyannulus_outer_pix)

# sky background estimate
sigma_clip = astropy.stats.SigmaClip (sigma=threshold, maxiters=maxiters)
apphot_sky_stats \
    = photutils.aperture.ApertureStats (data, apphot_annulus, \
                                        sigma_clip=sigma_clip)
skybg_per_pix     = apphot_sky_stats.mean
skybg_err_per_pix = apphot_sky_stats.std

# aperture photometry
```

```
phot_star    = photutils.aperture.aperture_photometry (data, apphot_aperture)
raw_flux     = phot_star['aperture_sum']
npix         = apphot_aperture.area
net_flux     = raw_flux - skybg_per_pix * npix
net_flux_err = numpy.sqrt (raw_flux + npix * skybg_err_per_pix**2)

# instrumental magnitude
instmag     = -2.5 * numpy.log10 (net_flux / exptime)
instmag_err = 2.5 / numpy.log (10) * net_flux_err / net_flux

# writing results into a file
with open (file_output, 'w') as fh:
    fh.write ("#\n")
    fh.write ("# Result of Aperture Photometry\n")
    fh.write ("#\n")
    fh.write ("#  Date/Time of Analysis\n")
    fh.write ("#   Date/Time = %s\n" % now)
    fh.write ("#\n")
    fh.write ("#  Input Parameters\n")
    fh.write ("#   FITS file                    = %s\n" % file_fits)
    fh.write ("#   RA                           = %f deg\n" % target_ra_deg)
    fh.write ("#   Dec                          = %f deg\n" % target_dec_deg)
    fh.write ("#   aperture radius              = %f in FWHM\n" \
              % aperture_radius_fwhm)
    fh.write ("#   inner sky annulus            = %f in FWHM\n" \
              % skyannulus_inner_fwhm)
    fh.write ("#   outer sky annulus            = %f in FWHM\n" \
              % skyannulus_outer_fwhm)
    fh.write ("#   half-width for centroid      = %f pixel\n" % halfwidth)
    fh.write ("#   threshold for sigma-clipping = %f in sigma\n" % threshold)
    fh.write ("#   number of max iterations     = %d\n" % maxiters)
    fh.write ("#   keyword for airmass          = %s\n" % keyword_airmass)
    fh.write ("#   keyword for exposure time    = %s\n" % keyword_exptime)
    fh.write ("#   keyword for filter name      = %s\n" % keyword_filter)
    fh.write ("#\n")
    fh.write ("#  Calculated and Measured Quantities\n")
    fh.write ("#   (init_x, init_y)             = (%f, %f)\n" \
              % (init_x, init_y) )
    fh.write ("#   (centroid_x, centroid_y)     = (%f, %f)\n" \
              % (x_centre + subframe_xmin, y_centre + subframe_ymin) )
    fh.write ("#   (centre_x, centre_y)         = (%f, %f)\n" \
              % (x_centre_psf, y_centre_psf) )
    fh.write ("#   FWHM of stellar PSF          = %f pixel\n" % fwhm)
    fh.write ("#   aperture radius              = %f pix\n" \
              % aperture_radius_pix)
    fh.write ("#   inner sky annulus            = %f pix\n" \
              % skyannulus_inner_pix)
    fh.write ("#   outer sky annulus            = %f pix\n" \
              % skyannulus_outer_pix)
    fh.write ("#   sky background level         = %f ADU per pixel\n" \
              % skybg_per_pix)
    fh.write ("#   net flux                     = %f ADU\n" % net_flux)
    fh.write ("#   net flux err                 = %f ADU\n" % net_flux_err)
    fh.write ("#   instrumental mag             = %f\n" % instmag)
    fh.write ("#   instrumental mag err         = %f\n" % instmag_err)
    fh.write ("#\n")
    fh.write ("#  Results\n")
    fh.write ("#   file, exptime, filter, centre_x, centre_y,\n"
              "#   net_flux, net_flux_err, instmag, instmag_err, airmass\n")
```

```
    fh.write ("%s %f %s %f %f %f %f %f %f %f\n" \
              % (file_fits, exptime, filter_name, x_centre_psf, y_centre_psf, \
                 net_flux, net_flux_err, instmag, instmag_err, airmass) )

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax = fig.add_subplot (111, projection=wcs)

# axes
ax.set_xlabel ('RA')
ax.set_ylabel ('Dec')
ax.set_xlim (int (x_centre_psf - skyannulus_outer_pix * 1.2),
             int (x_centre_psf + skyannulus_outer_pix * 1.2) )
ax.set_ylim (int (y_centre_psf - skyannulus_outer_pix * 1.2),
             int (y_centre_psf + skyannulus_outer_pix * 1.2) )

# plotting image
norm \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (data) )
im = ax.imshow (data, origin='lower', cmap=cmap, norm=norm)
fig.colorbar (im)

# making a circle to indicate the location of standard star
aperture = matplotlib.patches.Circle (xy=(x_centre_psf, y_centre_psf), \
                                      radius=aperture_radius_pix, \
                                      fill=False, color="red", linewidth=2)
annulus1 = matplotlib.patches.Circle (xy=(x_centre_psf, y_centre_psf), \
                                      radius=skyannulus_inner_pix, \
                                      fill=False, color="cyan", linewidth=2)
annulus2 = matplotlib.patches.Circle (xy=(x_centre_psf, y_centre_psf), \
                                      radius=skyannulus_outer_pix, \
                                      fill=False, color="cyan", linewidth=2)
# plotting location of standard star
ax.add_patch (aperture)
ax.add_patch (annulus1)
ax.add_patch (annulus2)

# invert Y-axis
ax.invert_yaxis ()

# saving file
fig.savefig (file_graphic, dpi=resolution)
```

Execute the script, and carry out aperture photometry of the star #025.

```
% chmod a+x advobs_202202_s14_05.py
% ./advobs_202202_s14_05.py -h
usage: advobs_202202_s14_05.py [-h] [-i INPUT] [-o OUTPUT] [-g GRAPHIC]
                               [-c {com,1dg,2dg}] [-p {2dg,2dm}] [-r RA]
                               [-d DEC] [-a APERTURE] [-w HALFWIDTH]
                               [-s1 SKYANNULUS1] [-s2 SKYANNULUS2]
                               [-t THRESHOLD] [-n MAXITERS]
                               [-e KEYWORD_EXPTIME] [-f KEYWORD_FILTER]
                               [-m KEYWORD_AIRMASS]
                               [-z {viridis,plasma,inferno,magma,cividis,binary,
gray,bone,pink,spring,summer,autumn,winter,cool,hot,copper,ocean,terrain,gnuplot
```

```
,cubehelix,jet,turbo}]
                                [-l RESOLUTION]

aperture photometry of a star at given RA and Dec

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        input FITS file name
  -o OUTPUT, --output OUTPUT
                        output data file name
  -g GRAPHIC, --graphic GRAPHIC
                        output graphic file name
  -c {com,1dg,2dg}, --centroid {com,1dg,2dg}
                        centroid measurement algorithm (default: 2dg)
  -p {2dg,2dm}, --psf {2dg,2dm}
                        PSF model [2dg=Gaussian, 2dm=Moffat] (default: 2dg)
  -r RA, --ra RA        RA in degree
  -d DEC, --dec DEC     Dec in degree
  -a APERTURE, --aperture APERTURE
                        aperture radius in FWHM (default: 2.0)
  -w HALFWIDTH, --halfwidth HALFWIDTH
                        half-width for centroid measurement (default: 20)
  -s1 SKYANNULUS1, --skyannulus1 SKYANNULUS1
                        inner sky annulus radius in FWHM (default: 4)
  -s2 SKYANNULUS2, --skyannulus2 SKYANNULUS2
                        outer sky annulus radius in FWHM (default: 7)
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for sigma-clipping in sigma (default: 4)
  -n MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 100)
  -e KEYWORD_EXPTIME, --keyword-exptime KEYWORD_EXPTIME
                        FITS keyword for exposure time (default: EXPTIME)
  -f KEYWORD_FILTER, --keyword-filter KEYWORD_FILTER
                        FITS keyword for filter name (default: FILTER)
  -m KEYWORD_AIRMASS, --keyword-airmass KEYWORD_AIRMASS
                        FITS keyword for airmass (default: AIRMASS)
  -z {viridis,plasma,inferno,magma,cividis,binary,gray,bone,pink,spring,summer,a
utumn,winter,cool,hot,copper,ocean,terrain,gnuplot,cubehelix,jet,turbo}, --cmap
{viridis,plasma,inferno,magma,cividis,binary,gray,bone,pink,spring,summer,autumn
,winter,cool,hot,copper,ocean,terrain,gnuplot,cubehelix,jet,turbo}
                        choice of colour map (default: bone)
  -l RESOLUTION, --resolution RESOLUTION
                        resolution in DPI (default: 450)

% ./advobs_202202_s14_05.py -i lot_20210214_0185_df.fits -o phot_0185_025.data \
? -g phot_0185_025.png -r 162.49926 -d -0.0433 -a 3.0
WARNING: FITSFixedWarning: RADECSYS= 'FK5 ' / Equatorial coordinate system
the RADECSYS keyword is deprecated, use RADESYSa. [astropy.wcs.wcs]
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to 59259.749086
 from DATE-OBS'. [astropy.wcs.wcs]
% ls -lF phot_0185_025.*
-rw-r--r--  1 daisuke  taiwan    1637 May  5 15:21 phot_0185_025.data
-rw-r--r--  1 daisuke  taiwan  238513 May  5 15:21 phot_0185_025.png
% cat phot_0185_025.data
#
# Result of Aperture Photometry
#
#  Date/Time of Analysis
```

```
#   Date/Time = 2022-05-05T15:21:46.992273
#
#  Input Parameters
#   FITS file                    = lot_20210214_0185_df.fits
#   RA                           = 162.499260 deg
#   Dec                          = -0.043300 deg
#   aperture radius              = 3.000000 in FWHM
#   inner sky annulus            = 4.000000 in FWHM
#   outer sky annulus            = 7.000000 in FWHM
#   half-width for centroid      = 20.000000 pixel
#   threshold for sigma-clipping = 4.000000 in sigma
#   number of max iterations     = 100
#   keyword for airmass          = AIRMASS
#   keyword for exposure time    = EXPTIME
#   keyword for filter name      = FILTER
#
#  Calculated and Measured Quantities
#   (init_x, init_y)             = (1456.862009, 1173.401432)
#   (centroid_x, centroid_y)     = (1457.429487, 1172.354975)
#   (centre_x, centre_y)         = (1457.429442, 1172.354845)
#   FWHM of stellar PSF          = 4.350898 pixel
#   aperture radius              = 13.052695 pix
#   inner sky annulus            = 17.403594 pix
#   outer sky annulus            = 30.456289 pix
#   sky background level         = 176.981055 ADU per pixel
#   net flux                     = 303229.589162 ADU
#   net flux err                 = 738.177042 ADU
#   instrumental mag             = -8.066248
#   instrumental mag err         = 0.002643
#
#  Results
#   file, exptime, filter, centre_x, centre_y,
#   net_flux, net_flux_err, instmag, instmag_err, airmass
lot_20210214_0185_df.fits 180.000000 gp_Astrodon_2019 1457.429442 1172.354845 30
3229.589162 738.177042 -8.066248 0.002643 1.122154
```

Check the sizes of aperture and sky annulus. (Fig. 7)

```
% feh -dF phot_0185_025.png
```

4.3 Measuring sky background level

4.3.1 Constructing a histogram of pixel values

Make a Python script to construct a histogram of pixel values of given FITS file.

Python Code 6: advobs202202_s14_06.py

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

#
# Time-stamp: <2022/05/05 15:48:18 (CST) daisuke>
#

# importing argparse module
import argparse
```

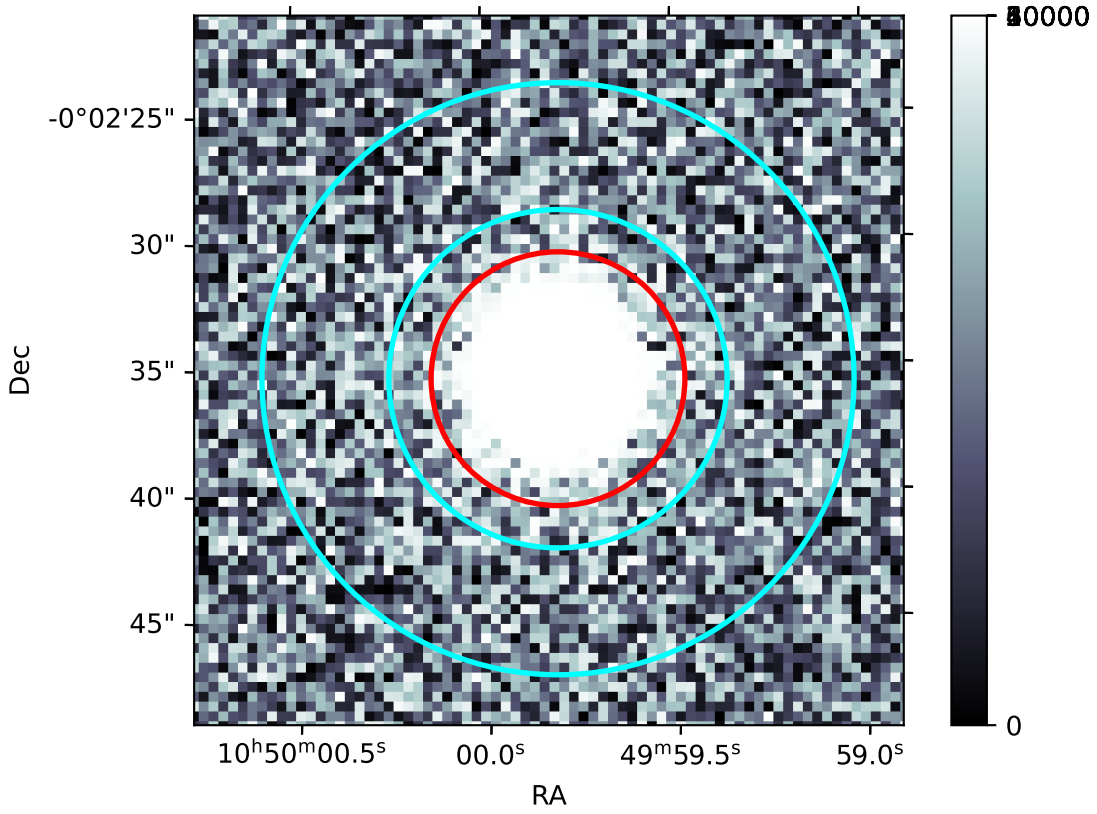

Figure 7: The sizes of aperture and sky annulus for the photometry of the standard star ID 25.

```
# importing sys module
import sys

# importing pathlib module
import pathlib

# importing astropy module
import astropy.io.fits

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# constructing parser object
desc   = "making a histogram"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input', default='', help='input FITS file')
parser.add_argument ('-o', '--output', default='', help='output file')
parser.add_argument ('-a', '--z1', type=float, default=0.0, \
                     help='minimum pixel value to plot (default: 0)')
parser.add_argument ('-b', '--z2', type=float, default=70000.0, \
                     help='maximum pixel value to plot (default: 70000)')
parser.add_argument ('-n', '--nbins', type=int, default=7000, \
                     help='number of bins (default: 7000)')
parser.add_argument ('-r', '--resolution', type=int, default=450, \
                     help='resolution of output image (default: 450 DPI)')

# parsing arguments
args = parser.parse_args ()

# input parameters
file_input  = args.input
file_output = args.output
z1          = args.z1
z2          = args.z2
nbins       = args.nbins
resolution  = args.resolution

# making a pathlib object
path_input  = pathlib.Path (file_input)
path_output = pathlib.Path (file_output)

# existence checks of files
if not (path_input.exists ()):
    # printing message
    print ("ERROR: input file does not exist.")
    # exit
    sys.exit ()
if (path_output.exists ()):
    # printing message
    print ("ERROR: output file exists.")
    # exit
    sys.exit ()

# check of input FITS file
if not (path_input.suffix == '.fits'):
    # printing message
```

```
    print ("ERROR: input file must be a FITS file.")
    print ("ERROR: given input file name = %s" % file_input)
    # exit
    sys.exit ()

# check of output file
if not ( (path_output.suffix == '.eps') or (path_output.suffix == '.pdf') \
         or (path_output.suffix == '.png') or (path_output.suffix == '.ps') ):
    # printing message
    print ("ERROR: output file must be either EPS, PDF, PNG, or PS.")
    print ("ERROR: given output file name = %s" % file_output)
    # exit
    sys.exit ()

# opening FITS file
with astropy.io.fits.open (file_input) as hdu_list:
    # reading image data
    data = hdu_list[0].data

# flattening data
data_flat = data.flatten ()

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax = fig.add_subplot (111)

# axes
ax.set_xlabel ("Pixel Value [ADU]")
ax.set_ylabel ("Number of pixels")

# plotting image
ax.set_xlim (z1, z2)
ax.hist (data_flat, bins=nbins, range=(z1, z2), histtype='bar', align='mid', \
         label=file_input)
ax.legend ()

# saving file
fig.savefig (file_output, dpi=resolution)
```

Run the script, and generate a histogram.

```
% chmod a+x advobs_202202_s14_06.py
% ./advobs_202202_s14_06.py -h
usage: advobs_202202_s14_06.py [-h] [-i INPUT] [-o OUTPUT] [-a Z1] [-b Z2]
                               [-n NBINS] [-r RESOLUTION]

making a histogram

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        input FITS file
  -o OUTPUT, --output OUTPUT
                        output file
  -a Z1, --z1 Z1        minimum pixel value to plot (default: 0)
  -b Z2, --z2 Z2        maximum pixel value to plot (default: 70000)
  -n NBINS, --nbins NBINS
```

```
                        number of bins (default: 7000)
  -r RESOLUTION, --resolution RESOLUTION
                        resolution of output image (default: 450 DPI)

% ./advobs_202202_s14_06.py -i lot_20210214_0185_df.fits -o hist_0185.png \
? -a 0 -b 300 -n 1000
% ls -lF hist_0185.png
-rw-r--r--  1 daisuke  taiwan  127437 May  5 15:56 hist_0185.png
```

Display the histogram. (Fig. 8)

```
% feh -dF hist_0185.png
```

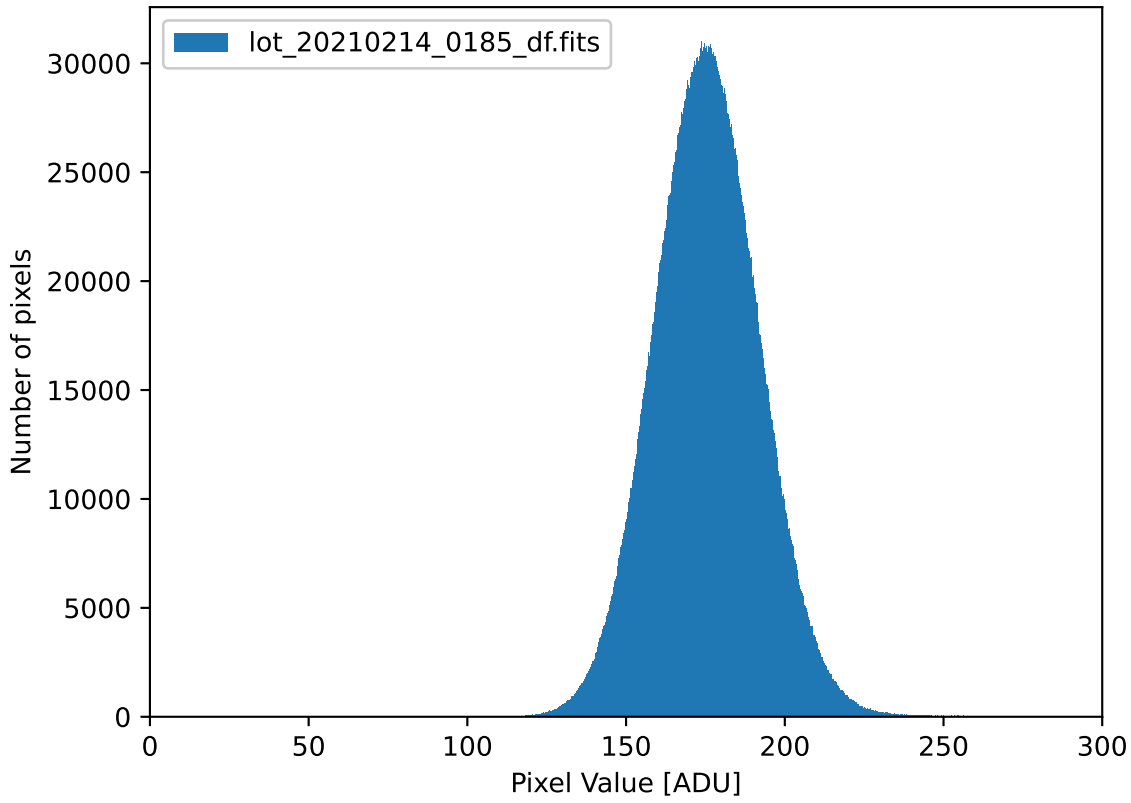

Figure 8: The histogram of the image `lot_20210214_0185_df.fits`.

Run the script again.

```
% ./advobs_202202_s14_06.py -i lot_20210214_0185_df.fits -o hist_0185_2.png \
? -a 100 -b 250 -n 1000
% ls -lF hist_0185*.png
-rw-r--r--  1 daisuke  taiwan  127437 May  5 15:56 hist_0185.png
-rw-r--r--  1 daisuke  taiwan  139685 May  5 16:01 hist_0185_2.png
```

Display newly created histogram. (Fig. 9)

```
% feh -dF hist_0185_2.png
```

The mean background level seems to be $\sim$ 175 ADU.

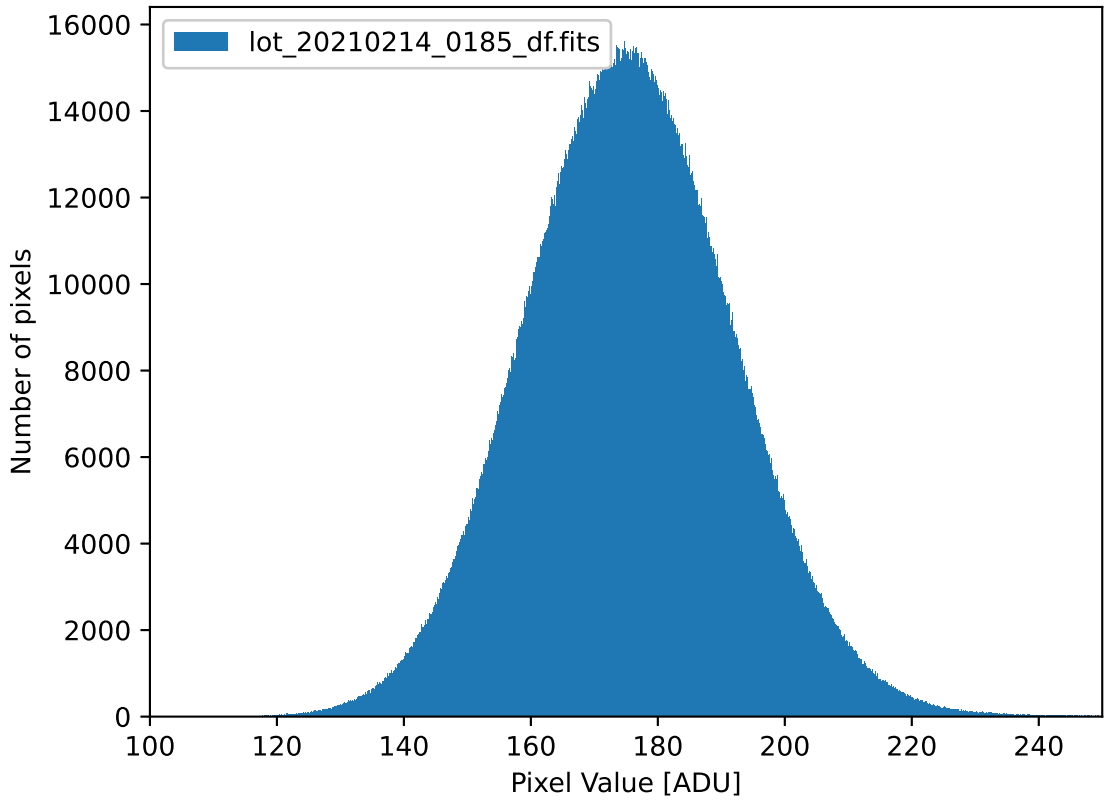

Figure 9: The histogram of the image `lot_20210214_0185_df.fits`.

4.3.2 Estimation of mode of distribution using empirical formula

Make a Python script to estimate mode of a given distribution using following empirical formula.

$$mode = 3 \times median - 2 \times mean \tag{1}$$

Python Code 7: advobs202202_s14_07.py

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

#
# Time-stamp: <2022/05/05 16:07:12 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
```

```
import pathlib

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

# constructing parser object
desc   = "estimating mode"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input', default='', help='input FITS file')
parser.add_argument ('-t', '--threshold', type=float, default=3.0, \
                     help='threshold for sigma-clipping in sigma (default: 3)')
parser.add_argument ('-n', '--maxiters', type=int, default=10, \
                     help='number of maximum iterations (default: 10)')

# parsing arguments
args = parser.parse_args ()

# input parameters
file_input = args.input
threshold  = args.threshold
maxiters   = args.maxiters

# making a pathlib object
path_input  = pathlib.Path (file_input)

# existence checks of file
if not (path_input.exists ()):
    # printing message
    print ("ERROR: input file does not exist.")
    # exit
    sys.exit ()

# check of input FITS file
if not (path_input.suffix == '.fits'):
    # printing message
    print ("ERROR: input file must be a FITS file.")
    print ("ERROR: given input file name = %s" % file_input)
    # exit
    sys.exit ()

# opening FITS file
with astropy.io.fits.open (file_input) as hdu_list:
    # reading image data
    data = hdu_list[0].data

# calculation of mean and median using sigma-clipping
mean, median, stddev \
    = astropy.stats.sigma_clipped_stats (data, sigma=threshold, \
                                         maxiters=maxiters, cenfunc='median')

# calculation of mode using empirical formula
mode = 3.0 * median - 2.0 * mean

print ("mean                          = %10.3f ADU" % mean)
print ("median                        = %10.3f ADU" % median)
print ("mode = 3 * median - 2 * mean = %10.3f ADU" % mode)
```

```
print ("stddev                          = %10.3f ADU" % stddev)
```

Execute the script.

```
% chmod a+x advobs_202202_s14_07.py
% ./advobs_202202_s14_07.py -h
usage: advobs_202202_s14_07.py [-h] [-i INPUT] [-t THRESHOLD] [-n MAXITERS]

estimating mode

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        input FITS file
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for sigma-clipping in sigma (default: 3)
  -n MAXITERS, --maxiters MAXITERS
                        number of maximum iterations (default: 10)

% ./advobs_202202_s14_07.py -i lot_20210214_0185_df.fits -t 4
mean                          =    175.566 ADU
median                        =    175.348 ADU
mode = 3 * median - 2 * mean =    174.911 ADU
stddev                        =     16.414 ADU
```

The sky background level is estimated to be 175 $\pm$ 16 ADU.

4.4 Calculating pixel scale

Make a Python script to calculate pixel scale of the image.

Python Code 8: advobs202202_s14_08.py

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

#
# Time-stamp: <2022/05/05 16:12:20 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits

# constructing parser object
desc   = "calculating pixel scale"
parser = argparse.ArgumentParser (description=desc)
```

```
# adding arguments
parser.add_argument ('-i', '--input', default='', help='input FITS file name')
parser.add_argument ('-x', '--keyword-pixsize-x', default='XPIXSZ', \
                     help='FITS keyword for pixel width in micron')
parser.add_argument ('-y', '--keyword-pixsize-y', default='YPIXSZ', \
                     help='FITS keyword for pixel height in micron')
parser.add_argument ('-l', '--keyword-focallength', default='FOCALLEN', \
                     help='FITS keyword for focal length in mm')

# parsing arguments
args = parser.parse_args ()

# input parameters
file_input          = args.input
keyword_pixelsize_x = args.keyword_pixsize_x
keyword_pixelsize_y = args.keyword_pixsize_y
keyword_focallength = args.keyword_focallength

# making a pathlib object
path_input  = pathlib.Path (file_input)

# existence checks of file
if not (path_input.exists ()):
    # printing message
    print ("ERROR: input file does not exist.")
    # exit
    sys.exit ()

# check of input FITS file
if not (path_input.suffix == '.fits'):
    # printing message
    print ("ERROR: input file must be a FITS file.")
    print ("ERROR: given input file name = %s" % file_input)
    # exit
    sys.exit ()

# opening FITS file
with astropy.io.fits.open (file_input) as hdu_list:
    # reading header information
    header = hdu_list[0].header

# pixel size and focal length
pixelsize_x_micron = header[keyword_pixelsize_x]
pixelsize_y_micron = header[keyword_pixelsize_y]
focallength_mm     = header[keyword_focallength]

# calculation of pixel scale
pixelscale_x_rad    = pixelsize_x_micron * 10**-6 / (focallength_mm * 10**-3)
pixelscale_y_rad    = pixelsize_y_micron * 10**-6 / (focallength_mm * 10**-3)
pixelscale_x_arcsec = pixelscale_x_rad * 180.0 / numpy.pi * 3600.0
pixelscale_y_arcsec = pixelscale_y_rad * 180.0 / numpy.pi * 3600.0

# printing result
print ("pixel scale along x-axis = %6.4f arcsec/pix" % pixelscale_x_arcsec)
print ("pixel scale along y-axis = %6.4f arcsec/pix" % pixelscale_y_arcsec)
```

Run the script, and show the result of pixel scale calculations.

```
% chmod a+x advobs_202202_s14_08.py
% ./advobs_202202_s14_08.py -h
usage: advobs_202202_s14_08.py [-h] [-i INPUT] [-x KEYWORD_PIXSIZE_X]
                               [-y KEYWORD_PIXSIZE_Y] [-l KEYWORD_FOCALLENGTH]

calculating pixel scale

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        input FITS file name
  -x KEYWORD_PIXSIZE_X, --keyword-pixsize-x KEYWORD_PIXSIZE_X
                        FITS keyword for pixel width in micron
  -y KEYWORD_PIXSIZE_Y, --keyword-pixsize-y KEYWORD_PIXSIZE_Y
                        FITS keyword for pixel height in micron
  -l KEYWORD_FOCALLENGTH, --keyword-focallength KEYWORD_FOCALLENGTH
                        FITS keyword for focal length in mm

% ./advobs_202202_s14_08.py -i lot_20210214_0185_df.fits
pixel scale along x-axis = 0.3855 arcsec/pix
pixel scale along y-axis = 0.3855 arcsec/pix
```

The pixel scale is 0.3855 arcsec/pix. Calculate the area of a pixel of the image projected on the sky.

```
% python3.9 -c 'print (0.3855**2, "arcsec^2")'
0.14861025 arcsec^2
```

Therefore, the area of a pixel of the image projected on the sky is 0.1486 arcsec^2.

4.5 Calculating sky background brightness in mag/arcsec^2

Make a Python script to calculate the sky background brightness in mag/arcsec^2.

Python Code 9: advobs202202_s14_09.py

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

#
# Time-stamp: <2022/05/05 16:18:57 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# constructing parser object
desc   = "calculating sky background brightness"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-b', '--skybg', type=float, default=-999.99, \
                     help='sky background level in ADU')
```

```
parser.add_argument ('-e', '--skybg-err', type=float, default=-999.99, \
                     help='error of sky background level in ADU')
parser.add_argument ('-s', '--star', type=float, default=-999.99, \
                     help='net flux of standard star in ADU')
parser.add_argument ('-m', '--mag', type=float, default=-999.99, \
                     help='apparent magnitude of standard star')
parser.add_argument ('-p', '--pixelscale', type=float, default=-999.99, \
                     help='pixel scale in arcsec/pix')

# parsing arguments
args = parser.parse_args ()

# input parameters
skybg_flux = args.skybg
skybg_err  = args.skybg_err
star_flux  = args.star
star_mag   = args.mag
pixelscale = args.pixelscale

# check of input parameters
if (skybg_flux < 0.0):
    print ("Something is wrong with sky background level in ADU.")
    sys.exit ()
if (skybg_err < 0.0):
    print ("Something is wrong with error of sky background level in ADU.")
    sys.exit ()
if (star_flux < 0.0):
    print ("Something is wrong with net flux of star in ADU.")
    sys.exit ()
if (star_mag < 0.0):
    print ("Something is wrong with magnitude of star.")
    sys.exit ()
if (pixelscale < 0.0):
    print ("Something is wrong with pixel scale in arcsec/pix.")
    sys.exit ()

# calculation of sky background brightness in mag/arcsec^2
skybg_flux_per_sqarcsec = skybg_flux / pixelscale**2
skybg_mag = star_mag - 2.5 * numpy.log10 (skybg_flux_per_sqarcsec / star_flux)
skybg_err = 2.5 / numpy.log (10) * skybg_err / skybg_flux

# printing result
print ("sky background brightness = %6.3f +/- %6.3f mag/arcsec^2" \
       % (skybg_mag, skybg_err) )
```

Run the script, and show the result of sky background brightness calculation.

```
% chmod a+x advobs_202202_s14_09.py
% ./advobs_202202_s14_09.py -h
usage: advobs_202202_s14_09.py [-h] [-b SKYBG] [-e SKYBG_ERR] [-s STAR]
                               [-m MAG] [-p PIXELSCALE]

calculating sky background brightness

optional arguments:
  -h, --help            show this help message and exit
  -b SKYBG, --skybg SKYBG
                        sky background level in ADU
```

```
  -e SKYBG_ERR, --skybg-err SKYBG_ERR
                        error of sky background level in ADU
  -s STAR, --star STAR  net flux of standard star in ADU
  -m MAG, --mag MAG     apparent magnitude of standard star
  -p PIXELSCALE, --pixelscale PIXELSCALE
                        pixel scale in arcsec/pix

% ./advobs_202202_s14_09.py -b 175 -e 16 -s 303230 -m 15.861 -p 0.3855
sky background brightness = 21.888 +/-  0.099 mag/arcsec^2
```

The sky background brightness in g'-band is estimated to be 21.9 ± 0.1 mag/arcsec^2.

4.6 The Moon on 14/Feb/2021

Use the software `XEphem` (Fig. 10) to check the Moon phase on 14/Feb/2021 and the position of the Moon on the sky at the time of the data acquisition.

- XEphem
  - `https://xephem.github.io/XEphem/Site/xephem.html`

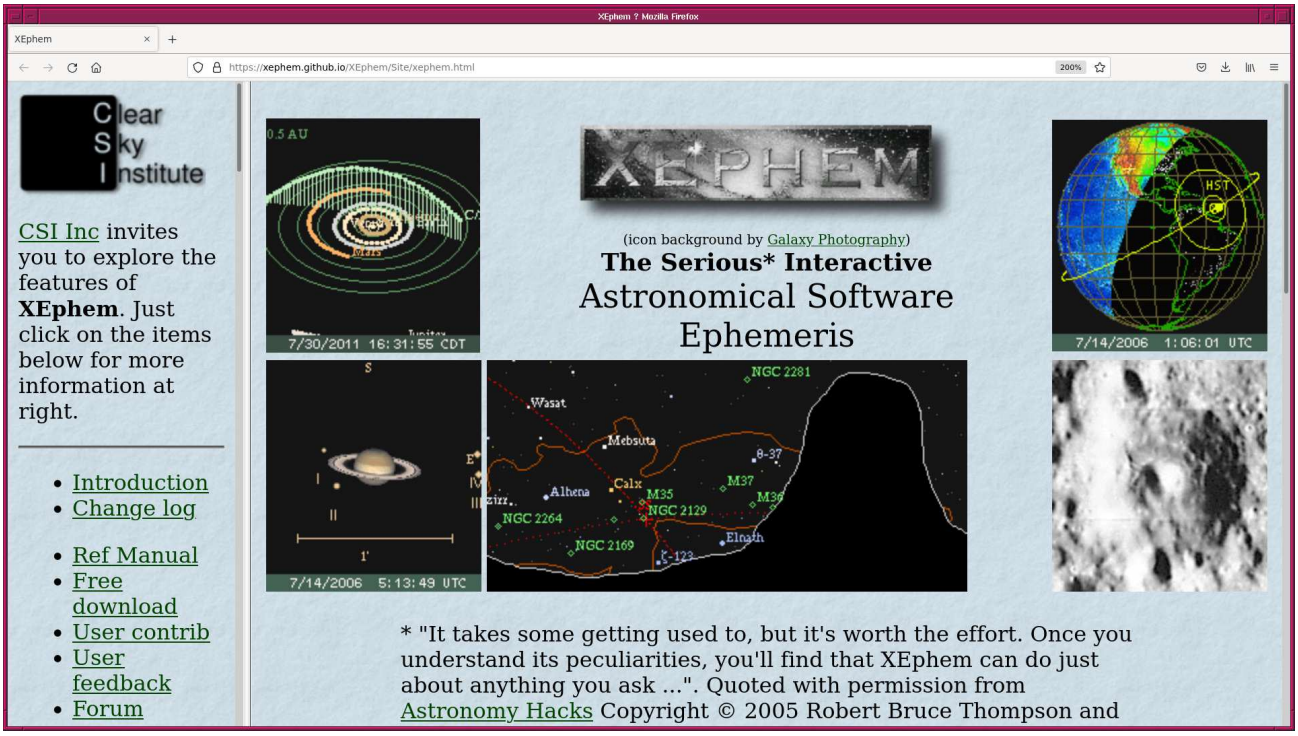

Figure 10: The official website of XEphem hosted on GitHub.

5 r'-band sky background brightness

Measure r'-band sky background brightness.

6 i'-band sky background brightness

Measure i'-band sky background brightness.

7 For your further reading

Read followings.

1. “Dictionaries” in “The Python Tutorial”
   - `https://docs.python.org/3/tutorial/datastructures.html#dictionaries`
2. “Flexible Image Transport System”
   - `https://fits.gsfc.nasa.gov/`
3. “FITS File Handling” of Astropy package
   - `https://docs.astropy.org/en/stable/io/fits/index.html`
4. “Astrostatistics Tools” of Astropy package
   - `https://docs.astropy.org/en/stable/stats/index.html`
5. matplotlib.pyplot.hist
   - `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html`
6. “Aperture Photometry” of Photutils package
   - `https://photutils.readthedocs.io/en/stable/aperture.html`
7. “Basic Photometry Techniques”
   - Da Costa
   - Astronomical CCD observing and reduction techniques, edited by Steve B. Howell. Published 1992 Astronomical Society of the Pacific, vol. 23, San Francisco, CA. ISBN 0-937707-42-4, LCCN 92-71172, p. 90.
   - `https://ui.adsabs.harvard.edu/abs/1992ASPC...23...90D/abstract`
8. “Estimating Lunar Phase Requirements”
   - Elias
   - 1994, NOAO Newsletter, Number 37.
   - `https://www.noao.edu/noao/noaonews/mar94/art20.html`
9. “A Model of the Brightness of Moonlight”
   - Krisciunas and Schaefer
   - 1991, PASP, 103, 1033.
   - `https://ui.adsabs.harvard.edu/abs/1991PASP..103.1033K/abstract`
10. “以M44疏散星團檢測鹿林前山之夜晚天光亮度”
    - 林宏欽 (Lin, Hung-Chin)
    - master thesis at National Central University
11. “Characteristics and Performance of the CCD Photometric System at Lulin Observatory”
    - Kinoshita et al.
    - 2005, ChJAA, 5, 315.
    - `https://ui.adsabs.harvard.edu/abs/2005ChJAA...5..315K/abstract`

8 Exercise

1. Sky background brightness
   - What are major contributions to night sky background brightness? Show references.
   - Use ADS (Astrophysics Data System, `https://ui.adsabs.harvard.edu/`) to search for papers reporting sky background brightness. Show sky background brightness in UBVRI and u'g'r'i'z' bands at major observatories in the world.
   - Use ADS (Astrophysics Data System, `https://ui.adsabs.harvard.edu/`) to search for papers reporting sky background brightness. Show sky background brightness at Lulin Observatory.
2. Estimate g'-band sky background brightness in mag/arcsec^2 using the FITS file `lot_20210214_0186_df.fits`. Show all the Python script you have written. Show the result of your estimate.
3. Estimate r'-band sky background brightness in mag/arcsec^2 using the FITS file `lot_20210214_0187_df.fits` or `lot_20210214_0188_df.fits`. Show all the Python script you have written. Show the result of your estimate.
4. Estimate i'-band sky background brightness in mag/arcsec^2 using the FITS file `lot_20210214_0189_df.fits` or `lot_20210214_0190_df.fits`. Show all the Python script you have written. Show the result of your estimate.
5. Use the formula in Krisciunas and Schaefer (1991) to estimate the night sky background brightness at zenith from your measurements in g', r', and i' images. Explain the method and results of your calculations.
6. Compare the results of your measurements with previously reported night sky background brightness at major astronomical sites in the world. Is the night sky background brightness at Lulin Observatory dark enough compared to those at major astronomical sites?
7. Search for publicly available archived data of photometric standard star field. Download an image of photometric standard star field. Analyse the image, and estimate the night sky background brightness in mag/arcsec^2. Show the result of your measurement, name of the observing site, and conditions of the observations.
8. Sky background brightness as a function of moon phase and wavelength
   - On dark nights, is the sky background brighter at shorter wavelength? Or, is it brighter at longer wavelength?
   - On bright nights, is the sky background brighter at shorter wavelength? Or, is it brighter at longer wavelength?
   - Do you find any trend between sky background brightness and wavelength?
   - How bright is the sky background in infrared wavelengths?