Advanced Astronomical Observations 2022
Session 03: Examining Bias Frames

Kinoshita Daisuke

03 March 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 examine bias frames. Bias frames are images of 0-sec exposure. Those data are taken when the shutter is closed. We read bias frames, extract data, visualise them, and calculate statistical values.

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 770 MB. It may take a while (∼ several minutes) to download the file depending on the network traffic condition.

- `https://s3b.astro.ncu.edu.tw/advobs_202202/data/data_s03.tar.xz`

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

```
% curl -k -o data_s03.tar.xz \
? https://s3b.astro.ncu.edu.tw/advobs_202202/data/data_s03.tar.xz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  772M  100  772M    0     0  1783k      0  0:07:23  0:07:23 --:--:-- 4520k
% ls -lF data_s03.tar.xz
-rw-r--r--  1 daisuke  taiwan  810503096 Mar  3 15:47 data_s03.tar.xz
% du -sm data_s03.tar.xz
774     data_s03.tar.xz
```

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. 285 FITS files should be extracted from the archive file.

```
% ls -l data_s03.tar.xz
-rw-r--r--  1 daisuke  taiwan  810503096 Mar  3 15:47 data_s03.tar.xz
% tar xJvf data_s03.tar.xz
x data_s03/
x data_s03/lot_20210212_0079.fits
x data_s03/lot_20210212_0080.fits
x data_s03/lot_20210212_0081.fits
x data_s03/lot_20210212_0082.fits
x data_s03/lot_20210212_0083.fits
x data_s03/lot_20210212_0084.fits
x data_s03/lot_20210212_0100.fits
x data_s03/lot_20210212_0101.fits
x data_s03/lot_20210212_0102.fits
x data_s03/lot_20210212_0112.fits

.....

x data_s03/lot_20210212_0809.fits
x data_s03/lot_20210212_0810.fits
x data_s03/lot_20210212_0811.fits
x data_s03/lot_20210212_0812.fits
x data_s03/lot_20210212_0813.fits
x data_s03/lot_20210212_0814.fits
x data_s03/lot_20210212_0815.fits
x data_s03/lot_20210212_0816.fits
x data_s03/lot_20210212_0817.fits
x data_s03/lot_20210212_0818.fits
% ls data_s03/ | head
lot_20210212_0079.fits
lot_20210212_0080.fits
lot_20210212_0081.fits
lot_20210212_0082.fits
lot_20210212_0083.fits
lot_20210212_0084.fits
lot_20210212_0100.fits
lot_20210212_0101.fits
lot_20210212_0102.fits
lot_20210212_0112.fits
% ls data_s03/*.fits | wc
     285     285    9120
```

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

```
% unxz -c data_s03.tar.xz | tar xvf -
x data_s03/
x data_s03/lot_20210212_0079.fits
x data_s03/lot_20210212_0080.fits
x data_s03/lot_20210212_0081.fits
x data_s03/lot_20210212_0082.fits
x data_s03/lot_20210212_0083.fits
x data_s03/lot_20210212_0084.fits
x data_s03/lot_20210212_0100.fits
x data_s03/lot_20210212_0101.fits
```

```
x data_s03/lot_20210212_0102.fits
x data_s03/lot_20210212_0112.fits

.....

x data_s03/lot_20210212_0809.fits
x data_s03/lot_20210212_0810.fits
x data_s03/lot_20210212_0811.fits
x data_s03/lot_20210212_0812.fits
x data_s03/lot_20210212_0813.fits
x data_s03/lot_20210212_0814.fits
x data_s03/lot_20210212_0815.fits
x data_s03/lot_20210212_0816.fits
x data_s03/lot_20210212_0817.fits
x data_s03/lot_20210212_0818.fits
```

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.

- XZ official website: `https://tukaani.org/xz/`

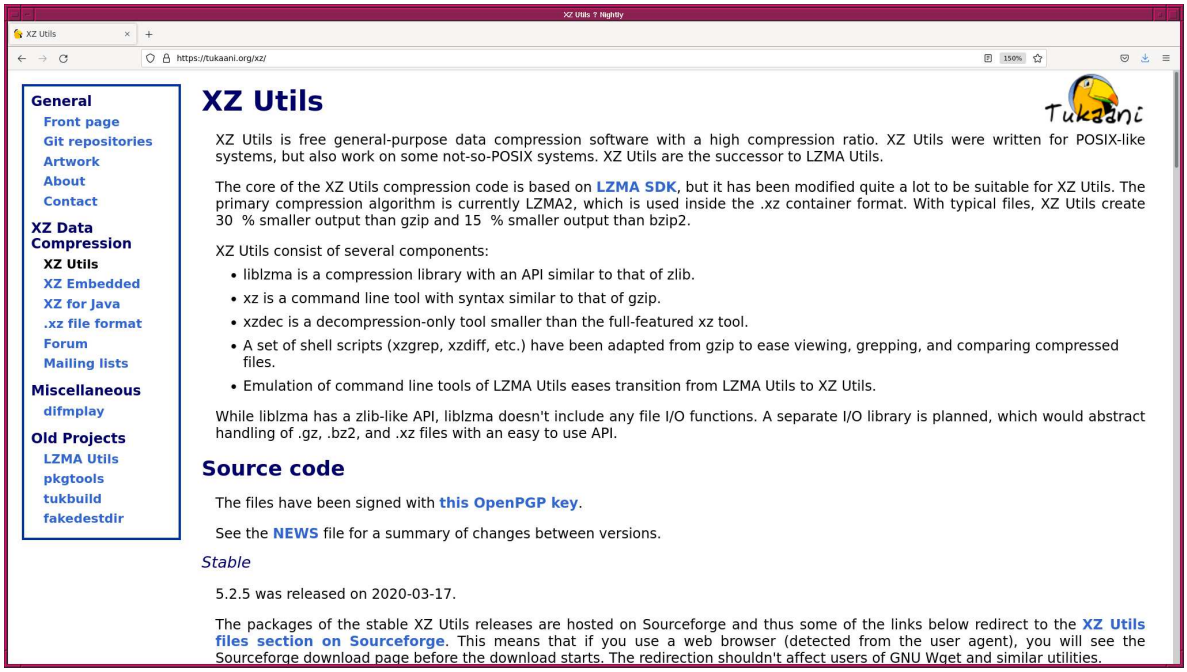

Figure 1: The website of XZ Utils.

If you are not familiar to pipes of Unix shells, try following. First, use the command `unxz` to decompress the file. Then, use the command `tar` to extract files.

```
% ls -l data_s03.tar.xz
-rw-r--r--  1 daisuke  taiwan  810503096 Mar  3 15:47 data_s03.tar.xz
% unxz -T 0 data_s03.tar.xz
% ls -l data_s03.tar
-rw-r--r--  1 daisuke  taiwan  2392797696 Mar  3 15:47 data_s03.tar
% tar xvf data_s03.tar
x data_s03/
x data_s03/lot_20210212_0079.fits
x data_s03/lot_20210212_0080.fits
```

```
x data_s03/lot_20210212_0081.fits
x data_s03/lot_20210212_0082.fits
x data_s03/lot_20210212_0083.fits
x data_s03/lot_20210212_0084.fits
x data_s03/lot_20210212_0100.fits
x data_s03/lot_20210212_0101.fits
x data_s03/lot_20210212_0102.fits
x data_s03/lot_20210212_0112.fits

.....

x data_s03/lot_20210212_0809.fits
x data_s03/lot_20210212_0810.fits
x data_s03/lot_20210212_0811.fits
x data_s03/lot_20210212_0812.fits
x data_s03/lot_20210212_0813.fits
x data_s03/lot_20210212_0814.fits
x data_s03/lot_20210212_0815.fits
x data_s03/lot_20210212_0816.fits
x data_s03/lot_20210212_0817.fits
x data_s03/lot_20210212_0818.fits
```

3 Checking the data type of images

Make a Python script to check the data type of FITS files.

Python Code 1: advobs202202_s03_01.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'Checking a keyword IMAGETYP'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('files', nargs='+', help='intput FITS files')

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

# input FITS file
list_files = args.files

# processing files
for file_fits in list_files:
    # if input file is not a FITS file, then skip
    if not (file_fits[-5:] == '.fits'):
        # printing a message
        print ("The file \"%s\" is not a FITS file!" % file_fits)
```

```
        # moving to next file
        continue

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

    # primary HDU
    hdu0 = hdu_list[0]

    # reading header
    header0 = hdu0.header

    # closing FITS file
    hdu_list.close ()

    # check of existence of IMAGETYP keyword
    if not ('IMAGETYP' in header0):
        print ("A keyword IMAGETYP does not exist in the file \"%s\"." \
               % file_fits)

    # check of existence of EXPTIME keyword
    if not ('EXPTIME' in header0):
        print ("A keyword EXPTIME does not exist in the file \"%s\"." \
               % file_fits)

    # printing values of IMAGETYP and EXPTIME keywords
    print ("%s : %s (integration time = %d sec)" \
           % (file_fits, header0['IMAGETYP'], header0['EXPTIME']) )
```

Execute the script, and check the data type of FITS files.

```
% chmod a+x advobs202202_s03_01.py
% ./advobs202202_s03_01.py -h
usage: advobs202202_s03_01.py [-h] files [files ...]

Checking a keyword IMAGETYP

positional arguments:
  files       intput FITS files

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

% ./advobs202202_s03_01.py data_s03/*.fits
data_s03/lot_20210212_0079.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0080.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0081.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0082.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0083.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0084.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0100.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0101.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0102.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0112.fits : BIAS (integration time = 0 sec)

.....

data_s03/lot_20210212_0809.fits : BIAS (integration time = 0 sec)
```

```
data_s03/lot_20210212_0810.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0811.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0812.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0813.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0814.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0815.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0816.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0817.fits : BIAS (integration time = 0 sec)
data_s03/lot_20210212_0818.fits : BIAS (integration time = 0 sec)
```

All the FITS files for this session are bias frames.

4 Reading a bias frame

Make a Python script to read a bias frame and show the shape of the data.

Python Code 2: advobs202202_s03_02.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'Reading image data from a FITS file'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input', default='test.fits', \
                     help='intput FITS file')

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

# input FITS file
file_input = args.input

# if input file is not a FITS file, then skip
if not (file_input[-5:] == '.fits'):
    # printing a message
    print ("Error: input file must be a FITS file!")
    # exit
    sys.exit ()

# file existence check using pathlib module
path_file_input = pathlib.Path (file_input)
if not (path_file_input.exists ()):
    # printing a message
    print ("Error: input file \"%s\" does not exist!" % file_input)
```

```
    # exit
    sys.exit ()

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

# primary HDU
hdu0 = hdu_list[0]

# reading header
header0 = hdu0.header

# reading data
data0 = hdu0.data

# closing FITS file
hdu_list.close ()

# printing shape of data
print ("object type        =", type (data0) )
print ("dimensions of data =", data0.ndim)
print ("shape of data      =", data0.shape)
print ("number of pixels   =", data0.size, "pixels")
print ("data type          =", data0.dtype)

# printing data
print ("pixel data:")
print (data0)
```

Choose one FITS file and execute the script.

```
% chmod a+x advobs202202_s03_02.py
% ./advobs202202_s03_02.py -h
usage: advobs202202_s03_02.py [-h] [-i INPUT]

Reading image data from a FITS file

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        intput FITS file

% ./advobs202202_s03_02.py -i data_s03/lot_20210212_0100.fits
object type        = <class 'numpy.ndarray'>
dimensions of data = 2
shape of data      = (2048, 2048)
number of pixels   = 4194304 pixels
data type          = uint16
pixel data:
[[606 606 615 ... 592 592 589]
 [610 593 592 ... 591 608 595]
 [591 590 587 ... 596 617 605]
 ...
 [593 608 587 ... 604 597 589]
 [594 592 586 ... 592 589 591]
 [604 595 587 ... 602 591 599]]
```

Choose the other FITS file, and try again.

```
% ./advobs202202_s03_02.py -i data_s03/lot_20210212_0150.fits
object type         = <class 'numpy.ndarray'>
dimensions of data = 2
shape of data       = (2048, 2048)
number of pixels    = 4194304 pixels
data type           = uint16
pixel data:
[[597 595 595 ... 600 597 603]
 [593 596 597 ... 603 606 608]
 [606 597 591 ... 587 592 592]
 ...
 [603 588 585 ... 602 608 578]
 [606 602 592 ... 595 597 594]
 [602 613 616 ... 595 588 600]]
```

The image data of a FITS file is stored in a Numpy array. It is a two-dimensional image, and the data type of each pixel value is 16-bit unsigned integer.

Try to specify a file extension other than "`.fits`".

```
% ./advobs202202_s03_02.py -i test.jpg
Error: input file must be a FITS file!
```

Try to specify a file name that does not exist.

```
% ./advobs202202_s03_02.py -i non_existence.fits
Error: input file "non_existence.fits" does not exist!
```

5 Visualising a bias frame

Make a Python script to convert a FITS file into a PNG image file. Here is an example.

Python Code 3: advobs202202_s03_03.py

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

# importing argparse module
import argparse

# 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

# construction of parser object
desc = 'Reading a FITS file and converting to a PNG file'
parser = argparse.ArgumentParser (description=desc)
```

```
# adding arguments
parser.add_argument ('-i', '--input', default='test.fits', \
                     help='intput FITS file')
parser.add_argument ('-o', '--output', default='test.png', \
                     help='output image file (EPS or PDF or PNG or PS)')
parser.add_argument ('-r', '--resolution', type=int, default=450, \
                     help='output image resolution (default: 450 dpi)')

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

# input FITS file
file_input  = args.input
file_output = args.output
resolution  = args.resolution

# if input file is not a FITS file, then skip
if not (file_input[-5:] == '.fits'):
    # printing a message
    print ("Error: input file must be a FITS file!")
    # exit
    sys.exit ()

# if output file is not either PNG, PDF, or PS, then skip
if not ( (file_output[-4:] == '.eps') or (file_output[-4:] == '.pdf') \
         or (file_output[-4:] == '.png') or (file_output[-3:] == '.ps') ):
    # printing a message
    print ("Error: output file must be a EPS or PDF or PNG or PS!")
    # exit
    sys.exit ()

# file existence check using pathlib module
path_file_input = pathlib.Path (file_input)
if not (path_file_input.exists ()):
    # printing a message
    print ("Error: input file \"%s\" does not exist!" % file_input)
    # exit
    sys.exit ()

# file existence check using pathlib module
path_file_output = pathlib.Path (file_output)
if (path_file_output.exists ()):
    # printing a message
    print ("Error: output file \"%s\" exists!" % file_output)
    # exit
    sys.exit ()

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

# primary HDU
hdu0 = hdu_list[0]

# reading header
header0 = hdu0.header

# reading data
data0 = hdu0.data
```

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

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

# axes
ax.set_title (file_input)
ax.set_xlabel ('X [pixel]')
ax.set_ylabel ('Y [pixel]')

# plotting image
im = ax.imshow (data0, origin='lower', cmap='inferno')
fig.colorbar (im)

# saving file
print ("converting image: %s ==> %s" % (file_input, file_output) )
fig.savefig (file_output, dpi=resolution)
```

Run the script, and convert a FITS file into a PNG file.

```
% chmod a+x advobs202202_s03_03.py
% ./advobs202202_s03_03.py -h
usage: advobs202202_s03_03.py [-h] [-i INPUT] [-o OUTPUT] [-r RESOLUTION]

Reading a FITS file and converting to a PNG file

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        intput FITS file
  -o OUTPUT, --output OUTPUT
                        output image file (EPS or PDF or PNG or PS)
  -r RESOLUTION, --resolution RESOLUTION
                        output image resolution (default: 450 dpi)

% ./advobs202202_s03_03.py -i data_s03/lot_20210212_0100.fits -o b0100.png
converting image: data_s03/lot_20210212_0100.fits ==> b0100.png
% ls -lF b0100.png
-rw-r--r--  1 daisuke  taiwan  7388575 Mar  3 16:16 b0100.png
% file b0100.png
b0100.png: PNG image data, 2880 x 2160, 8-bit/color RGBA, non-interlaced
```

Display the PNG image using your favourite image viewer software. (Fig. 2) For example, a command named "`feh`" can be used for this purpose.

```
% feh -dF b0100.png
```

If you do not have a command "`feh`" on your computer, and are willing to install it, visit the official web site of "`feh`". (Fig. 3)

- FEH official website: `https://feh.finalrewind.org/`

If you prefer to use GIMP, then try following.

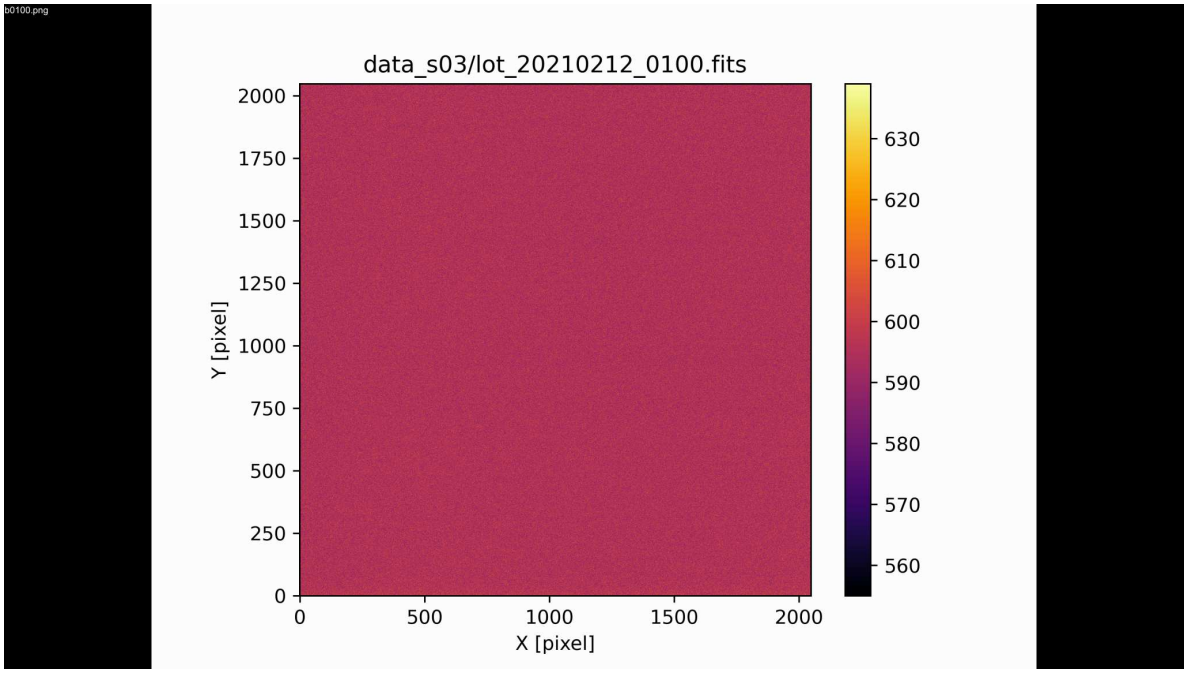

Figure 2: A visualised bias frame.

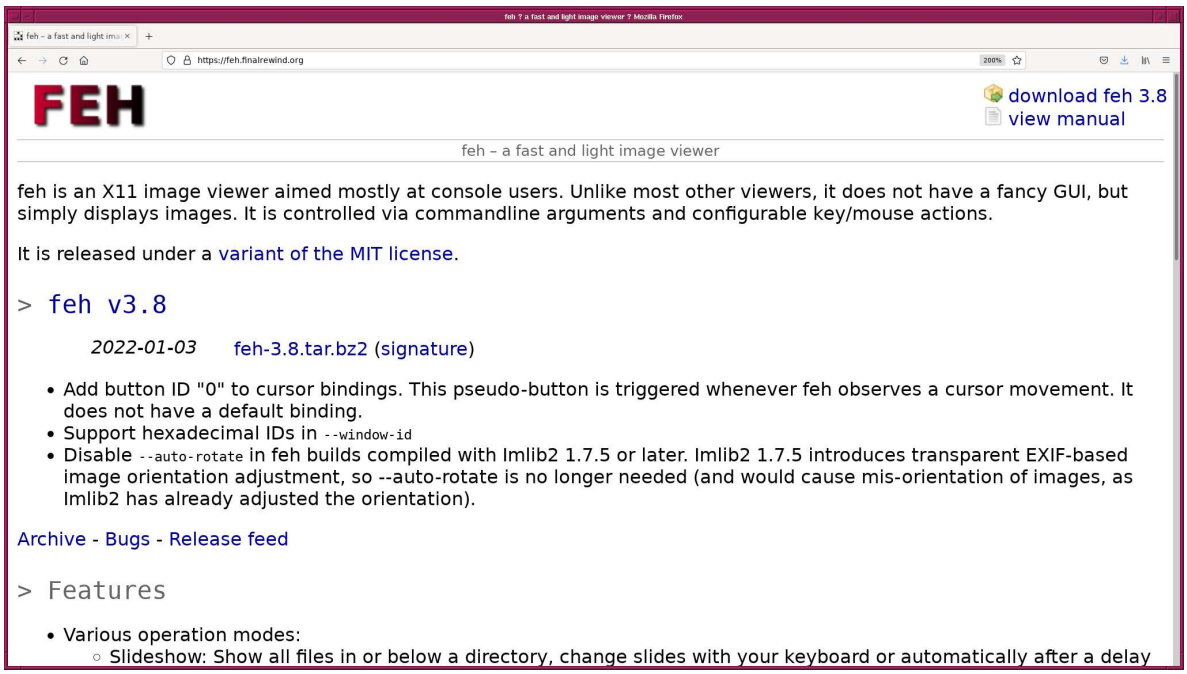

Figure 3: The official web page of the image viewer "`feh`".

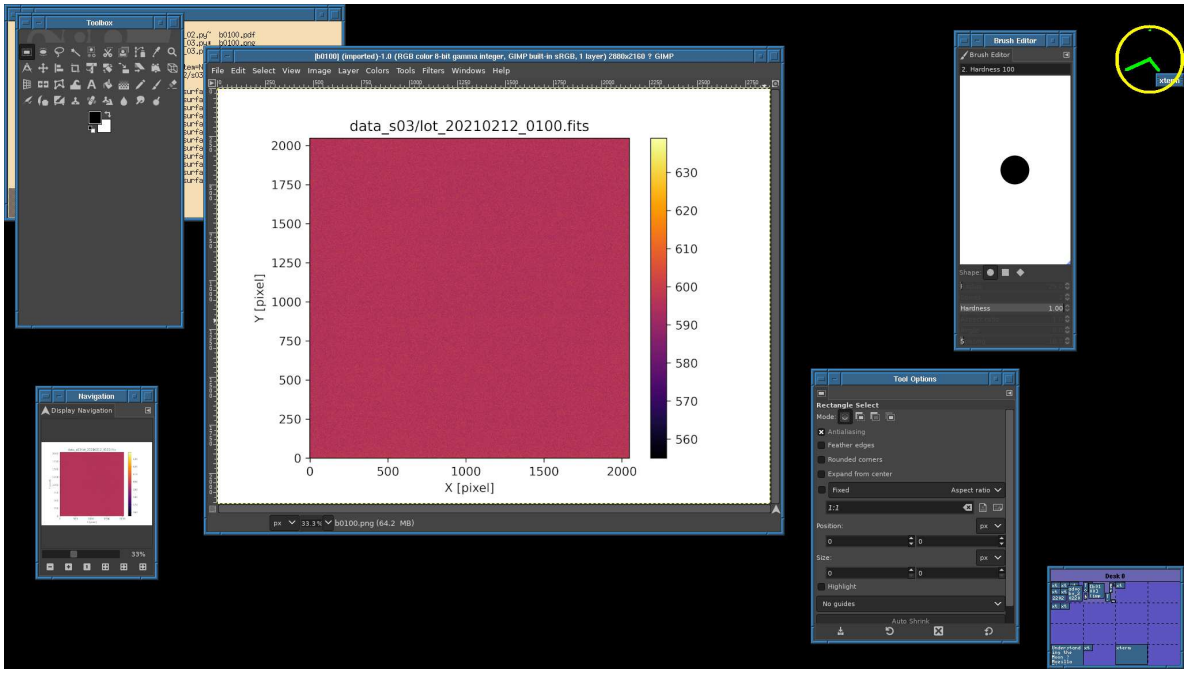

Figure 4: The PNG file "`b0100.pdf`" displayed using the command "`gimp`".

```
% gimp b0100.png &
```

If you do not have a command "`gimp`" on your computer, and are willing to install it, visit the official web site of GIMP. (Fig. 5)

- GIMP official website: `https://www.gimp.org/`

If you prefer PDF format, try following.

```
% ./advobs202202_s03_03.py -i data_s03/lot_20210212_0100.fits -o b0100.pdf
converting image: data_s03/lot_20210212_0100.fits ==> b0100.pdf
% ls -lF b0100.*
-rw-r--r--  1 daisuke  taiwan  2113699 Mar  3 16:35 b0100.pdf
-rw-r--r--  1 daisuke  taiwan  7388575 Mar  3 16:16 b0100.png
```

Use your favourite PDF viewer software to display the file "`b0100.pdf`". Here is an example. (Fig. 6)

```
% okular b0100.pdf
```

6 Calculating a simple mean of a bias frame

Make a Python script to read a bias frame and calculate a simple mean.

Python Code 4: ao2021_s03_04.py

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

# importing argparse module
import argparse
```

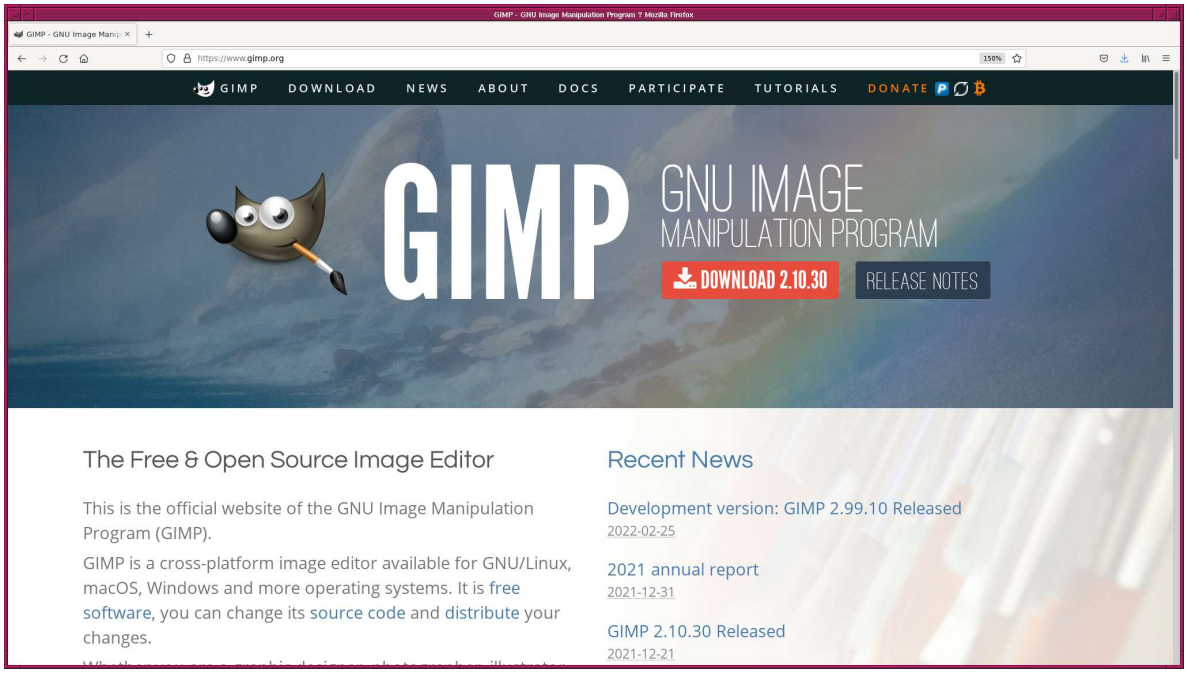

Figure 5: The official web page of the image manipulation program "GIMP".

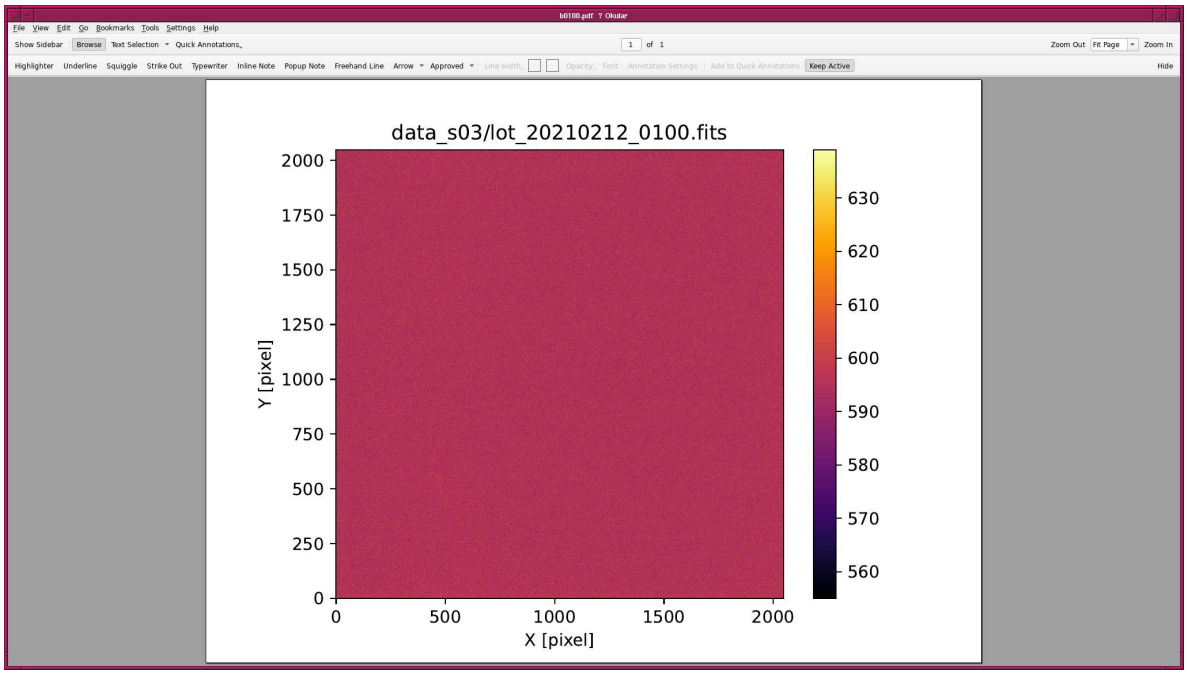

Figure 6: The PDF file "`b0100.pdf`" displayed using the command "`okular`".

```
# importing sys module
import sys

# importing pathlib module
import pathlib

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'Reading a FITS file and calculating a mean'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input', default='test.fits', \
                     help='intput FITS file')

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

# input FITS file
file_input = args.input

# if input file is not a FITS file, then skip
if not (file_input[-5:] == '.fits'):
    # printing a message
    print ("Error: input file must be a FITS file!")
    # exit
    sys.exit ()

# file existence check using pathlib module
path_file_input = pathlib.Path (file_input)
if not (path_file_input.exists ()):
    # printing a message
    print ("Error: input file \"%s\" does not exist!" % file_input)
    # exit
    sys.exit ()

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

# primary HDU
hdu0 = hdu_list[0]

# reading header
header0 = hdu0.header

# reading data
data0 = hdu0.data

# closing FITS file
hdu_list.close ()

# a variable for a sum of all the pixel values
total = 0.0
# total number of pixels
n = data0.size
```

```
# adding all the pixel values
for i in range (len (data0) ):
    for j in range (len (data0[i]) ):
        # adding pixel value data0[i][j] to the total
        total += data0[i][j]

# calculation of a simple mean
mean = total / n

# printing result
print ("sum of all the pixel values = %f" % total)
print ("number of pixels            = %d" % n)
print ("mean                        = %f" % mean)
```

Execute the script, and calculate a mean of a bias frame.

```
% chmod a+x advobs202202_s03_04.py
% ./advobs202202_s03_04.py -h
usage: advobs202202_s03_04.py [-h] [-i INPUT]

Reading a FITS file and calculating a mean

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        intput FITS file

% ./advobs202202_s03_04.py -i data_s03/lot_20210212_0100.fits
sum of all the pixel values = 2496782470.000000
number of pixels            = 4194304
mean                        = 595.279329
```

The mean of all the pixel values of the image `lot_20210212_0100.fits` is 595.28 ADU.
Try one more.

```
% ./advobs202202_s03_04.py -i data_s03/lot_20210212_0150.fits
sum of all the pixel values = 2497340078.000000
number of pixels            = 4194304
mean                        = 595.412273
```

If you specify a non-existent FITS file name, then the script stops.

```
% ./advobs202202_s03_04.py -i non_existent_file.fits
Error: input file "non_existent_file.fits" does not exist!
```

7 Calculating mean, median, max, min, and stddev

In addition to a simple mean, calculate median, maximum value, minimum value, and standard deviation.

Python Code 5: advobs202202_s03_05.py

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

# importing argparse module
import argparse
```

```
# importing sys module
import sys

# import pathlib module
import pathlib

# importing math module
import math

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'Reading a FITS file and calculating statistical values'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input', default='test.fits', \
                     help='intput FITS file')

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

# input FITS file
file_input = args.input

# if input file is not a FITS file, then skip
if not (file_input[-5:] == '.fits'):
    # printing a message
    print ("Error: input file must be a FITS file!")
    # exit
    sys.exit ()

# file existence check using pathlib module
path_file_input = pathlib.Path (file_input)
if not (path_file_input.exists ()):
    # printing a message
    print ("Error: input file \"%s\" does not exist!" % file_input)
    # exit
    sys.exit ()

# file name
filename = path_file_input.name

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

# primary HDU
hdu0 = hdu_list[0]

# reading header
header0 = hdu0.header

# reading data
data0 = hdu0.data

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

```
# initial value for maximum value
v_max = -9.99 * 10**10
# initial value for minimum value
v_min = +9.99 * 10**10
# a variable for a sum of all the pixel values
total = 0.0
# a variable for a square of sum of all the pixel values
total_sq = 0.0
# total number of pixels
n = data0.size
# a 1-dim. list for data
data0_1d = []

# calculating statistical values
for i in range (len (data0) ):
    for j in range (len (data0[i]) ):
        # for calculation of a mean
        total += data0[i][j]
        # for calculation of a variance
        total_sq += data0[i][j]**2
        # for maximum value
        if (data0[i][j] > v_max):
            v_max = data0[i][j]
        # for minimum value
        if (data0[i][j] < v_min):
            v_min = data0[i][j]
        # making 1-dim. list
        data0_1d.append (data0[i][j])

# mean
mean   = total / n
# variance
var    = total_sq / n - mean**2
# standard deviation
stddev = math.sqrt (var)

# sorting
data0_1d_sorted = sorted (data0_1d)
# median
if (n % 2 == 0):
    median = (data0_1d_sorted[int (n / 2) - 1] \
              + data0_1d_sorted[int (n / 2)]) / 2.0
else:
    median = data0_1d_sorted[int (n / 2)]

# printing result
print ("# file name, mean, median, stddev, min, max")
print ("%s    %f %f %f %f %f" \
       % (filename, mean, median, stddev, v_min, v_max) )
```

Run the script.

```
% chmod a+x advobs202202_s03_05.py
% ./advobs202202_s03_05.py -i data_s03/lot_20210212_0100.fits
lot_20210212_0100.fits    595.279329 595.000000 7.913900 555.000000 639.000000
```

The mean, median, stddev, min, and max for the file `lot_20210212_0100.fits` are 595.28, 595.00, 7.91, 555.00, and 639.00 respectively.

Modify the script to accept multiple FITS files.

Python Code 6: advobs202202_s03_06.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# import pathlib module
import pathlib

# importing math module
import math

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'Reading FITS files and calculating statistical values'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('files', nargs='+', help='intput FITS files')

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

# input FITS file
list_files = args.files

# printing header
print ("# file name, mean, median, stddev, min, max")

# processing files one-by-one
for file_input in list_files:
    # if input file is not a FITS file, then skip
    if not (file_input[-5:] == '.fits'):
        # printing a message
        print ("# Error: input file must be a FITS file!")
        # skipping to next file
        continue

    # file existence check using pathlib module
    path_file_input = pathlib.Path (file_input)
    if not (path_file_input.exists ()):
        # printing a message
        print ("# Error: input file \"%s\" does not exist!" % file_input)
        # skipping to next file
        continue

    # file name
    filename = path_file_input.name
```

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

    # primary HDU
    hdu0 = hdu_list[0]

    # reading header
    header0 = hdu0.header

    # reading data
    data0 = hdu0.data

    # closing FITS file
    hdu_list.close ()

    # initial value for maximum value
    v_max = -9.99 * 10**10
    # initial value for minimum value
    v_min = +9.99 * 10**10
    # a variable for a sum of all the pixel values
    total = 0.0
    # a variable for a square of sum of all the pixel values
    total_sq = 0.0
    # total number of pixels
    n = data0.size
    # a 1-dim. list for data
    data0_1d = []

    # calculating statistical values
    for i in range (len (data0) ):
        for j in range (len (data0[i]) ):
            # for calculation of a mean
            total += data0[i][j]
            # for calculation of a variance
            total_sq += data0[i][j]**2
            # for maximum value
            if (data0[i][j] > v_max):
                v_max = data0[i][j]
            # for minimum value
            if (data0[i][j] < v_min):
                v_min = data0[i][j]
            # making 1-dim. list
            data0_1d.append (data0[i][j])

    # mean
    mean   = total / n
    # variance
    var    = total_sq / n - mean**2
    # standard deviation
    stddev = math.sqrt (var)

    # sorting
    data0_1d_sorted = sorted (data0_1d)
    # median
    if (n % 2 == 0):
        median = (data0_1d_sorted[int (n / 2) - 1] \
                  + data0_1d_sorted[int (n / 2)]) / 2.0
    else:
        median = data0_1d_sorted[int (n / 2)]
```

```
    # printing result
    print ("%s  %10.3f %10.3f %10.3f %10.3f %10.3f" \
           % (filename, mean, median, stddev, v_min, v_max) )
```

Execute the script.

```
% chmod a+x advobs202202_s03_06.py
% ./advobs202202_s03_06.py -h
usage: advobs202202_s03_06.py [-h] files [files ...]

Reading FITS files and calculating statistical values

positional arguments:
  files       intput FITS files

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

% ./advobs202202_s03_06.py data_s03/lot_20210212_010?.fits
# file name, mean, median, stddev, min, max
lot_20210212_0100.fits     595.279    595.000      7.914    555.000    639.000
lot_20210212_0101.fits     595.583    596.000      7.912    558.000    635.000
lot_20210212_0102.fits     595.342    595.000      7.971    552.000   1700.000
```

Try to include non-existent file names.

```
% ./advobs202202_s03_06.py a.fits data_s03/lot_20210212_011?.fits b.fits
# file name, mean, median, stddev, min, max
# Error: input file "a.fits" does not exist!
lot_20210212_0112.fits     595.434    595.000      7.933    557.000    636.000
lot_20210212_0113.fits     595.093    595.000      8.349    553.000   4142.000
lot_20210212_0114.fits     595.789    596.000      7.930    555.000   1328.000
# Error: input file "b.fits" does not exist!
```

8 Calculating statistical values using Numpy

Use Numpy and calculate statistical values.

Python Code 7: advobs202202_s03_07.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# import pathlib module
import pathlib

# importing numpy module
import numpy

# importing astropy module
```

```
import astropy.io.fits

# construction of parser object
desc = 'Reading FITS files and calculating statistical values using Numpy'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('files', nargs='+', help='intput FITS files')

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

# input FITS file
list_files = args.files

# printing header
print ("# file name, mean, median, stddev, min, max")

# processing files one-by-one
for file_input in list_files:
    # if input file is not a FITS file, then skip
    if not (file_input[-5:] == '.fits'):
        # printing a message
        print ("# Error: input file must be a FITS file!")
        # skipping to next file
        continue

    # file existence check using pathlib module
    path_file_input = pathlib.Path (file_input)
    if not (path_file_input.exists ()):
        # printing a message
        print ("# Error: input file \"%s\" does not exist!" % file_input)
        # skipping to next file
        continue

    # file name
    filename = path_file_input.name

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

    # primary HDU
    hdu0 = hdu_list[0]

    # reading header
    header0 = hdu0.header

    # reading data
    data0 = hdu0.data

    # closing FITS file
    hdu_list.close ()

    # calculating statistical values
    mean   = numpy.mean (data0)
    median = numpy.median (data0)
    stddev = numpy.std (data0)
    v_min  = numpy.amin (data0)
    v_max  = numpy.amax (data0)
```

```
    # printing result
    print ("%s  %10.3f %10.3f %10.3f %10.3f %10.3f" \
           % (filename, mean, median, stddev, v_min, v_max) )
```

Run the script and do the calculation.

```
% chmod a+x advobs202202_s03_07.py
% ./advobs202202_s03_07.py -h
usage: advobs202202_s03_07.py [-h] files [files ...]

Reading FITS files and calculating statistical values using Numpy

positional arguments:
  files       intput FITS files

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

% ./advobs202202_s03_07.py data_s03/lot_20210212_012?.fits
# file name, mean, median, stddev, min, max
lot_20210212_0124.fits     595.696    596.000      7.871    555.000   1030.000
lot_20210212_0125.fits     595.928    596.000      7.863    550.000    638.000
lot_20210212_0126.fits     595.331    595.000      7.877    554.000   1201.000
```

Compare the results of calculations by `advobs202202_s03_06.py` and `advobs202202_s03_07.py`.

```
% ./advobs202202_s03_06.py data_s03/lot_20210212_0100.fits > stat_0100_1.data
% ./advobs202202_s03_07.py data_s03/lot_20210212_0100.fits > stat_0100_2.data
% cat stat_0100_1.data
# file name, mean, median, stddev, min, max
lot_20210212_0100.fits     595.279    595.000      7.914    555.000    639.000
% cat stat_0100_2.data
# file name, mean, median, stddev, min, max
lot_20210212_0100.fits     595.279    595.000      7.914    555.000    639.000
% diff stat_0100_?.data
```

Two Python scripts give the exactly the same results.
Try the script for the file `lot_20210212_0084.fits`.

```
% ./advobs202202_s03_07.py data_s03/lot_20210212_0084.fits
# file name, mean, median, stddev, min, max
lot_20210212_0084.fits     595.102    595.000      8.044    555.000   2202.000
```

A pixel of the image `lot_20210212_0084.fits` has an anomalously high pixel value of 2202.00. We should think of more robust way to calculate a mean by rejecting outliers.

9 Constructing a histogram

Make a Python script to construct a histogram of pixel values of a bias frame.

Python Code 8: advobs202202_s03_08.py

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

# 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

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

# construction of parser object
desc = 'Reading a FITS file and constructing a histogram'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input', default='test.fits', \
                     help='intput FITS file')
parser.add_argument ('-o', '--output', default='test.png', \
                     help='output image file')
parser.add_argument ('-a', '--min', type=float, default=400.0, \
                     help='minimum value for histogram')
parser.add_argument ('-b', '--max', type=float, default=800.0, \
                     help='maximum value for histogram')
parser.add_argument ('-w', '--width', type=int, default=1.0, \
                     help='width of a bin for histogram')

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

# input FITS file
file_input  = args.input
file_output = args.output

# parameters
a     = args.min
b     = args.max
width = args.width
nbin  = int ( (b - a) / width ) + 1

# if input file is not a FITS file, then skip
if not (file_input[-5:] == '.fits'):
    # printing a message
    print ("Error: input file must be a FITS file!")
    # exit
    sys.exit ()

# if output file is not either PNG, PDF, or PS, then skip
```

```
if not ( (file_output[-4:] == '.eps') or (file_output[-4:] == '.pdf') \
         or (file_output[-4:] == '.png') or (file_output[-3:] == '.ps') ):
    # printing a message
    print ("Error: output file must be a PNG or PDF or PS!")
    # exit
    sys.exit ()

# existence check of input file using pathlib module
path_file_input = pathlib.Path (file_input)
if not (path_file_input.exists ()):
    # printing a message
    print ("Error: input file \"%s\" does not exist!" % file_input)
    # exit
    sys.exit ()

# existence check of output file using pathlib module
path_file_output = pathlib.Path (file_output)
if (path_file_output.exists ()):
    # printing a message
    print ("Error: output file \"%s\" exists!" % file_output)
    # exit
    sys.exit ()

# if a >= b, then skip
if (a >= b):
    # printing a message
    print ("maximum value \"a\" must be greater than minimum value \"b\".")
    # exit
    sys.exit ()

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

# primary HDU
hdu0 = hdu_list[0]

# reading header
header0 = hdu0.header

# reading data
data0 = hdu0.data

# closing FITS file
hdu_list.close ()

# initialisation of Numpy arrays for histogram
histogram_x = numpy.linspace (a, b, nbin)
histogram_y = numpy.zeros (nbin, dtype='int64')

# making a histogram
for i in range ( len (data0) ):
    for j in range ( len (data0[i]) ):
        # skipping data if it is outside the range [a, b]
        if ( (data0[i][j] > b) or (data0[i][j] < a) ):
            continue
        # counting
        histogram_y[int ( (data0[i][j] - a) / width)] += 1

# making objects "fig" and "ax"
```

```
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax = fig.add_subplot (111)

# labels
label_x = 'Pixel Value [ADU]'
label_y = 'Number of Pixels'
ax.set_xlabel (label_x)
ax.set_ylabel (label_y)

# plotting histogram
ax.bar (histogram_x, histogram_y, width, edgecolor='black', \
        linewidth=0.3, align='edge', label='Pixel values')
ax.legend ()

# saving the figure to a file
fig.savefig (file_output, dpi=450)
```

Run the script.

```
% chmod a+x advobs202202_s03_08.py
% ./advobs202202_s03_08.py -h
usage: advobs202202_s03_08.py [-h] [-i INPUT] [-o OUTPUT] [-a MIN] [-b MAX]
                              [-w WIDTH]

Reading a FITS file and constructing a histogram

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        intput FITS file
  -o OUTPUT, --output OUTPUT
                        output image file
  -a MIN, --min MIN     minimum value for histogram
  -b MAX, --max MAX     maximum value for histogram
  -w WIDTH, --width WIDTH
                        width of a bin for histogram

% ./advobs202202_s03_08.py -i data_s03/lot_20210212_0100.fits -o hist_0100.png
% ls -l hist_0100.png
-rw-r--r--  1 daisuke  taiwan  108469 Mar  3 22:39 hist_0100.png
```

Display the image. (Fig. 7)

```
% feh -dF hist_0100.png
```

Adjust the range, and run the script again.

```
% ./advobs202202_s03_08.py -i data_s03/lot_20210212_0100.fits \
? -o hist_0100_2.png -a 550 -b 650
% ls -l hist_0100_2.png
-rw-r--r--  1 daisuke  taiwan  108720 Mar  3 22:45 hist_0100_2.png
```

Display the image. (Fig. 8)

```
% feh -dF hist_0100_2.png
```

Almost all the pixels have pixel values between $(\bar{x} - 30)$ and $(\bar{x} + 30)$ ADU which roughly correspond $\bar{x} \pm 4\sigma$.

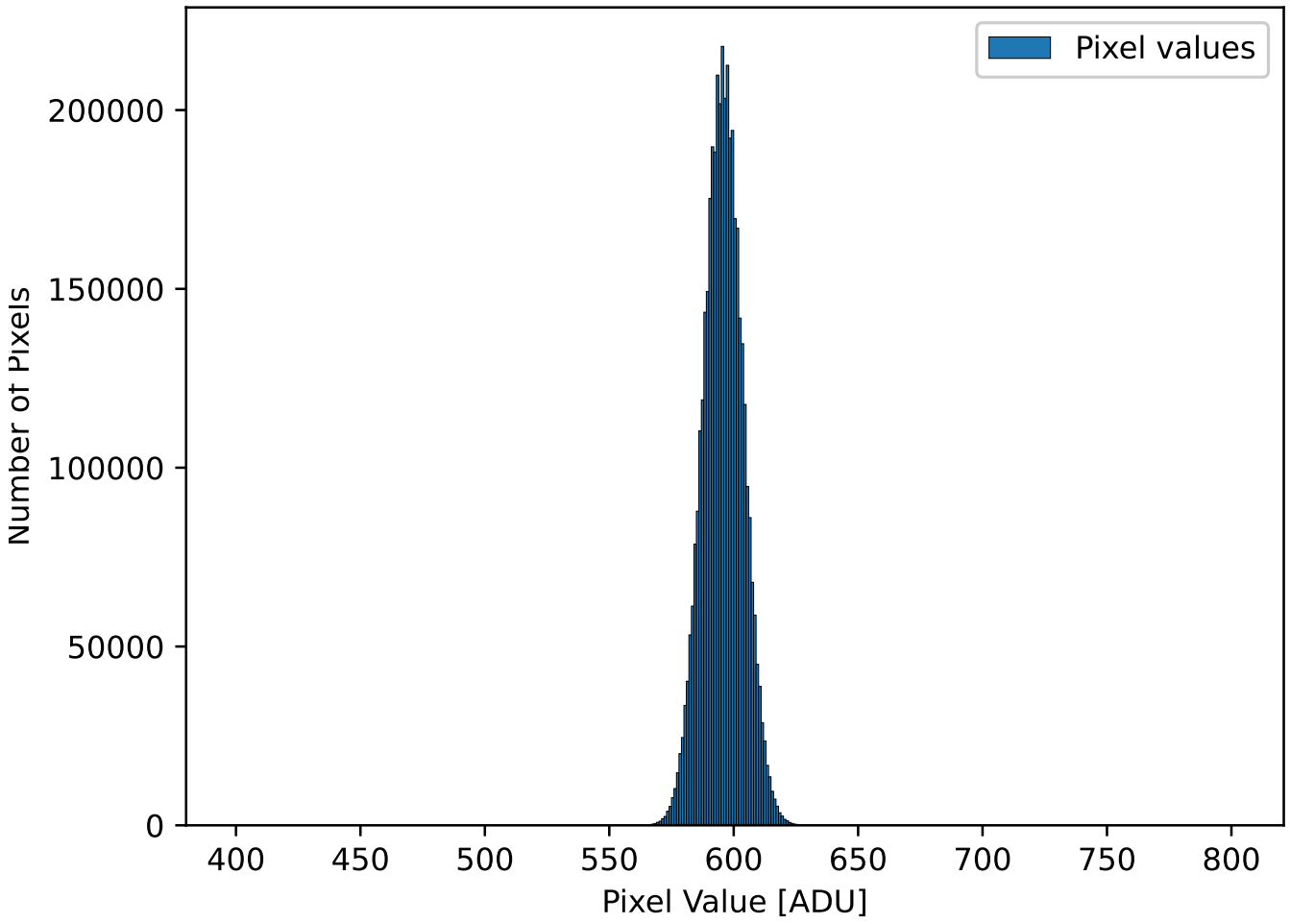

Figure 7: The histogram of pixel values of a bias frame `lot_20210212_0100.fits`.

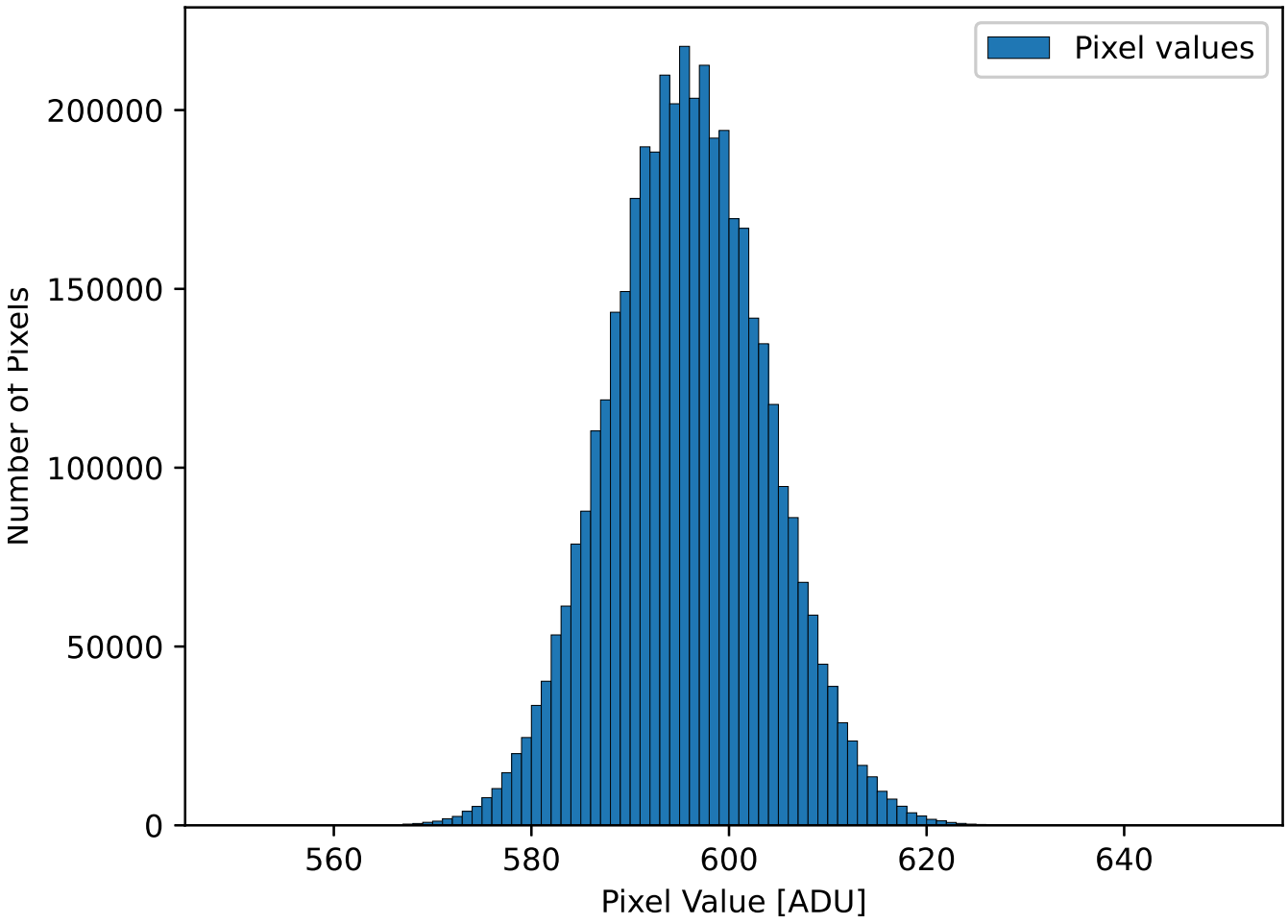

Figure 8: The updated histogram of pixel values of a bias frame `lot_20210212_0100.fits`.

10 Calculating a mean using sigma clipping

More robust estimate of a mean is possible by using sigma clipping algorithm. Make a Python script to calculate a mean using sigma clipping algorithm.

Python Code 9: advobs202202_s03_09.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing math module
import math

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'Calculating statistical values using sigma clipping'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('files', nargs='+', help='intput FITS files')
parser.add_argument ('-s', '--sigma', type=float, default=5.0, \
                     help='factor for sigma clipping')

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

# input FITS file
list_files = args.files
nsigma     = args.sigma

# processing files one-by-one
for file_input in list_files:
    # if input file is not a FITS file, then skip
    if not (file_input[-5:] == '.fits'):
        # printing a message
        print ("Error: input file must be a FITS file!")
        # exit
        sys.exit ()

    # file existence check using pathlib module
    path_file_input = pathlib.Path (file_input)
    if not (path_file_input.exists ()):
        # printing a message
        print ("Error: input file \"%s\" does not exist!" % file_input)
        # exit
        sys.exit ()

    # file name
    filename = path_file_input.name
```

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

    # primary HDU
    hdu0 = hdu_list[0]

    # reading header
    header0 = hdu0.header

    # reading data
    data0 = hdu0.data

    # closing FITS file
    hdu_list.close ()

    # initial value for maximum value
    v_max         = -9.99 * 10**10
    v_max_clipped = -9.99 * 10**10
    # initial value for minimum value
    v_min         = +9.99 * 10**10
    v_min_clipped = +9.99 * 10**10
    # a variable for a sum of all the pixel values
    total         = 0.0
    total_clipped = 0.0
    # a variable for a square of sum of all the pixel values
    total_sq         = 0.0
    total_sq_clipped = 0.0
    # total number of pixels
    n = data0.size

    # calculating statistical values
    for row in data0:
        for x in row:
            # for calculation of a mean
            total += x
            # for calculation of a variance
            total_sq += x**2
            # for maximum value
            if (x > v_max):
                v_max = x
            # for minimum value
            if (x < v_min):
                v_min = x

    # mean
    mean   = total / n
    # variance
    var    = total_sq / n - mean**2
    # standard deviation
    stddev = math.sqrt (var)

    # lists for rejected and accepted values
    rejected = []
    accepted = []

    # calculating statistical values
    for row in data0:
        for x in row:
            # if the pixel value is outside of
```

```
            # [mean-nsigma*stddev,mean+nsigma*stddev], then reject
            if ( (x > mean + nsigma * stddev) or (x < mean - nsigma * stddev) ):
                # appending the pixel value to the list "rejected"
                rejected.append (x)
            else:
                # appending the pixel value to the list "accepted"
                accepted.append (x)
                # for calculation of a mean
                total_clipped += x
                # for calculation of a variance
                total_sq_clipped += x**2
                # for maximum value
                if (x > v_max_clipped):
                    v_max_clipped = x
                # for minimum value
                if (x < v_min_clipped):
                    v_min_clipped = x

    # numbers of accepted and rejected pixels
    n_accepted = len (accepted)
    n_rejected = len (rejected)

    # mean
    mean_clipped   = total_clipped / n_accepted
    # variance
    var_clipped    = total_sq_clipped / n_accepted - mean_clipped**2
    # standard deviation
    stddev_clipped = math.sqrt (var_clipped)

    # printing result
    print ("before clipping:")
    print ("  %s  %10.3f %10.3f %10.3f %10.3f" \
           % (filename, mean, stddev, v_min, v_max) )
    print ("after clipping:")
    print ("  %s  %10.3f %10.3f %10.3f %10.3f" \
           % (filename, mean_clipped, stddev_clipped, \
              v_min_clipped, v_max_clipped) )
    print ("results of sigma-clipping:")
    print ("  number of accepted pixels = %10d" % n_accepted)
    print ("  number of rejected pixels = %10d" % n_rejected)
    print ("  rejected pixel values     =", sorted (rejected) )
```

Execute the script.

```
% chmod a+x advobs202202_s03_09.py
% ./advobs202202_s03_09.py -h
usage: advobs202202_s03_09.py [-h] [-s SIGMA] files [files ...]

Calculating statistical values using sigma clipping

positional arguments:
  files                 intput FITS files

optional arguments:
  -h, --help            show this help message and exit
  -s SIGMA, --sigma SIGMA
                        factor for sigma clipping
```

```
% ./advobs202202_s03_09.py data_s03/lot_20210212_0084.fits
before clipping:
  lot_20210212_0084.fits     595.102      8.044    555.000   2202.000
after clipping:
  lot_20210212_0084.fits     595.101      8.004    555.000    635.000
results of sigma-clipping:
  number of accepted pixels =    4194297
  number of rejected pixels =          7
  rejected pixel values     = [637, 640, 646, 691, 712, 942, 2202]
```

Calculate sigma-clipped mean for the file `lot_20210212_0102.fits`.

```
% ./advobs202202_s03_09.py data_s03/lot_20210212_0102.fits
before clipping:
  lot_20210212_0102.fits     595.342      7.971    552.000   1700.000
after clipping:
  lot_20210212_0102.fits     595.341      7.925    556.000    634.000
results of sigma-clipping:
  number of accepted pixels =    4194291
  number of rejected pixels =         13
  rejected pixel values     = [552, 554, 637, 693, 709, 716, 720, 863, 1054, 106
0, 1191, 1535, 1700]
```

Sigma clipping should be an iterative process. Make a Python script to calculate sigma-clipped mean and standard deviation using iterative process.

Python Code 10: advobs202202_s03_10.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing math module
import math

# importing numpy module
import numpy

# importing astropy module
import astropy.io.fits

# construction of parser object
desc = 'Calculating statistical values using sigma clipping with iterations'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('files', nargs='+', help='intput FITS files')
parser.add_argument ('-s', '--sigma', type=float, default=5.0, \
                     help='factor for sigma clipping (default: 5.0)')
parser.add_argument ('-n', '--nmaxiter', type=int, default=10, \
```

```
                    help='number of maximum iterations (default: 10)')

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

# input FITS file
list_files = args.files
nsigma     = args.sigma
nmaxiter   = args.nmaxiter

# processing files one-by-one
for file_input in list_files:
    # if input file is not a FITS file, then skip
    if not (file_input[-5:] == '.fits'):
        # printing a message
        print ("Error: input file must be a FITS file!")
        # exit
        sys.exit ()

    # file existence check using pathlib module
    path_file_input = pathlib.Path (file_input)
    if not (path_file_input.exists ()):
        # printing a message
        print ("Error: input file \"%s\" does not exist!" % file_input)
        # exit
        sys.exit ()

    # file name
    filename = path_file_input.name

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

    # primary HDU
    hdu0 = hdu_list[0]

    # reading header
    header0 = hdu0.header

    # reading data
    data0 = hdu0.data

    # closing FITS file
    hdu_list.close ()

    # initial value for maximum value
    v_max          = -9.99 * 10**10
    # initial value for minimum value
    v_min          = +9.99 * 10**10
    # a variable for a sum of all the pixel values
    total          = 0.0
    # a variable for a square of sum of all the pixel values
    total_sq          = 0.0
    # total number of pixels
    n = data0.size

    # flattening numpy array
    fdata0 = data0.flatten ()
```

```
    # calculating statistical values
    for x in fdata0:
        # for calculation of a mean
        total += x
        # for calculation of a variance
        total_sq += x**2
        # for maximum value
        if (x > v_max):
            v_max = x
        # for minimum value
        if (x < v_min):
            v_min = x

    # mean
    mean   = total / n
    # variance
    var    = total_sq / n - mean**2
    # standard deviation
    stddev = math.sqrt (var)

    # lists for rejected and accepted values
    rejected = []
    accepted = fdata0

    # number of iteration
    niter = 0

    # mean and stddev
    mean_clipped   = mean
    stddev_clipped = stddev

    while (niter < nmaxiter):
        # printing status
        print ("# iteration #%03d..." % (niter + 1) )

        # initialising parameters
        total_clipped    = 0.0
        total_sq_clipped = 0.0
        v_max_clipped    = -9.99 * 10**10
        v_min_clipped    = +9.99 * 10**10

        # number of rejected pixels before sigma-clipping
        n_rejected_prev = len (rejected)

        # temporary list of accepted pixels
        tmp_accepted = []

        # calculating statistical values
        for i in range ( len (accepted) ):
            # if the pixel value is outside of
            # [mean-nsigma*stddev,mean+nsigma*stddev], then reject
            if ( (accepted[i] > mean_clipped + nsigma * stddev_clipped) \
                 or (accepted[i] < mean_clipped - nsigma * stddev_clipped) ):
                # appending the pixel value to the list "rejected"
                rejected.append (accepted[i])
            else:
                # appending the pixel value to the list "tmp_accepted"
                tmp_accepted.append (accepted[i])
                # for calculation of a mean
```

```
                total_clipped += accepted[i]
                # for calculation of a variance
                total_sq_clipped += accepted[i]**2
                # for maximum value
                if (accepted[i] > v_max_clipped):
                    v_max_clipped = accepted[i]
                # for minimum value
                if (accepted[i] < v_min_clipped):
                    v_min_clipped = accepted[i]

        # new list of accepted pixels
        accepted = tmp_accepted

        # numbers of accepted and rejected pixels
        n_accepted = len (accepted)
        n_rejected = len (rejected)

        # printing status
        print ("#  n_accepted = %d, n_rejected = %d" \
               % (n_accepted, n_rejected) )

        # mean
        mean_clipped   = total_clipped / n_accepted
        # variance
        var_clipped    = total_sq_clipped / n_accepted - mean_clipped**2
        # standard deviation
        stddev_clipped = math.sqrt (var_clipped)

        # checking whether or not leaving from iteration
        if (n_rejected == n_rejected_prev):
            break
        else:
            niter += 1

    # printing result
    print ("before clipping:")
    print ("  %s  %10.3f %10.3f %10.3f %10.3f" \
           % (filename, mean, stddev, v_min, v_max) )
    print ("after clipping:")
    print ("  %s  %10.3f %10.3f %10.3f %10.3f" \
           % (filename, mean_clipped, stddev_clipped, \
              v_min_clipped, v_max_clipped) )
    print ("results of sigma-clipping:")
    print ("  number of accepted pixels = %10d" % n_accepted)
    print ("  number of rejected pixels = %10d" % n_rejected)
    print ("  rejected pixel values     =", sorted (rejected) )
```

Execute the script.

```
% chmod a+x advobs202202_s03_10.py
% ./advobs202202_s03_10.py -h
usage: advobs202202_s03_10.py [-h] [-s SIGMA] [-n NMAXITER] files [files ...]

Calculating statistical values using sigma clipping with iterations

positional arguments:
  files                 intput FITS files
```

```
optional arguments:
  -h, --help            show this help message and exit
  -s SIGMA, --sigma SIGMA
                        factor for sigma clipping (default: 5.0)
  -n NMAXITER, --nmaxiter NMAXITER
                        number of maximum iterations (default: 10)

% ./advobs202202_s03_10.py data_s03/lot_20210212_0102.fits
# iteration #001...
#  n_accepted = 4194291, n_rejected = 13
# iteration #002...
#  n_accepted = 4194291, n_rejected = 13
before clipping:
  lot_20210212_0102.fits     595.342      7.971    552.000   1700.000
after clipping:
  lot_20210212_0102.fits     595.341      7.925    556.000    634.000
results of sigma-clipping:
  number of accepted pixels =    4194291
  number of rejected pixels =         13
  rejected pixel values     = [552, 554, 637, 693, 709, 716, 720, 863, 1054, 106
0, 1191, 1535, 1700]
```

Try one more.

```
% ./advobs202202_s03_10.py data_s03/lot_20210212_0113.fits
# iteration #001...
#  n_accepted = 4194256, n_rejected = 48
# iteration #002...
#  n_accepted = 4194255, n_rejected = 49
# iteration #003...
#  n_accepted = 4194255, n_rejected = 49
before clipping:
  lot_20210212_0113.fits     595.093      8.349    553.000   4142.000
after clipping:
  lot_20210212_0113.fits     595.087      7.931    557.000    634.000
results of sigma-clipping:
  number of accepted pixels =    4194255
  number of rejected pixels =         49
  rejected pixel values     = [553, 635, 639, 640, 646, 651, 658, 661, 662, 662,
 674, 675, 676, 683, 690, 692, 693, 709, 722, 747, 748, 759, 850, 851, 851, 903,
 943, 946, 967, 1061, 1069, 1098, 1117, 1153, 1195, 1201, 1246, 1313, 1396, 1428
, 1440, 1453, 1458, 1538, 1658, 1846, 2079, 2278, 4142]
```

11 Sigma clipping using Astropy

Use Astropy to carry out sigma clipping.

Python Code 11: advobs202202_s03_11.py

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

# 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
import astropy.stats

# construction of parser object
desc = 'Calculating statistical values using sigma clipping by Astropy'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('files', nargs='+', help='intput FITS file')
parser.add_argument ('-s', '--sigma', type=float, default=5.0, \
                     help='factor for sigma clipping')
parser.add_argument ('-n', '--nmaxiter', type=int, default=10, \
                     help='maximum number of iterations')

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

# input FITS file
list_files = args.files
nsigma     = args.sigma
nmaxiter   = args.nmaxiter

# processing file one-by-one
for file_input in list_files:
    # if input file is not a FITS file, then skip
    if not (file_input[-5:] == '.fits'):
        # printing a message
        print ("# Error: input file must be a FITS file!")
        # skipping to next
        continue

    # file existence check using pathlib module
    path_file_input = pathlib.Path (file_input)
    if not (path_file_input.exists ()):
        # printing a message
        print ("# Error: input file \"%s\" does not exist!" % file_input)
        # skipping to next
        continue

    # file name
    filename = path_file_input.name

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

    # primary HDU
    hdu0 = hdu_list[0]

    # reading header
    header0 = hdu0.header

    # reading data
```

```
    data0 = hdu0.data

    # closing FITS file
    hdu_list.close ()

    # simple mean
    mean   = numpy.mean (data0)
    stddev = numpy.std (data0)
    v_min  = numpy.amin (data0)
    v_max  = numpy.amax (data0)

    # sigma clipped mean
    data0_sigclip  = astropy.stats.sigma_clip (data0, sigma=nsigma, \
                                               maxiters=nmaxiter, masked=False)
    mean_sigclip   = numpy.mean (data0_sigclip)
    stddev_sigclip = numpy.std (data0_sigclip)
    v_min_sigclip  = numpy.amin (data0_sigclip)
    v_max_sigclip  = numpy.amax (data0_sigclip)

    # printing result
    print ("before clipping:")
    print ("  %s  %10.3f %10.3f %10.3f %10.3f" \
           % (filename, mean, stddev, v_min, v_max) )
    print ("after clipping:")
    print ("  %s  %10.3f %10.3f %10.3f %10.3f" \
           % (filename, mean_sigclip, stddev_sigclip, \
              v_min_sigclip, v_max_sigclip) )
    print ("size of data0              = %d" % data0.size)
    print ("size of data0_sigclip      = %d" % data0_sigclip.size)
    print ("number of rejected pixels = %d" \
           % (data0.size - data0_sigclip.size) )
```

Run the script, and compare the results with those calculated by the previous script.

```
% chmod a+x advobs202202_s03_11.py
% ./advobs202202_s03_11.py -h
usage: advobs202202_s03_11.py [-h] [-s SIGMA] [-n NMAXITER] files [files ...]

Calculating statistical values using sigma clipping by Astropy

positional arguments:
  files                 intput FITS file

optional arguments:
  -h, --help            show this help message and exit
  -s SIGMA, --sigma SIGMA
                        factor for sigma clipping
  -n NMAXITER, --nmaxiter NMAXITER
                        maximum number of iterations

% ./advobs202202_s03_11.py data_s03/lot_20210212_0102.fits
before clipping:
  lot_20210212_0102.fits     595.342      7.971    552.000   1700.000
after clipping:
  lot_20210212_0102.fits     595.341      7.925    556.000    634.000
size of data0              = 4194304
size of data0_sigclip      = 4194291
number of rejected pixels = 13
```

Try one more.

```
% ./advobs202202_s03_11.py data_s03/lot_20210212_0113.fits
before clipping:
  lot_20210212_0113.fits     595.093      8.349    553.000   4142.000
after clipping:
  lot_20210212_0113.fits     595.087      7.931    557.000    634.000
size of data0              = 4194304
size of data0_sigclip      = 4194255
number of rejected pixels = 49
```

To learn more about sigma-clipping using Astropy, read following official document of Astropy.

- Astrostatistics Tools: `https://docs.astropy.org/en/stable/stats/` (Fig. 9)

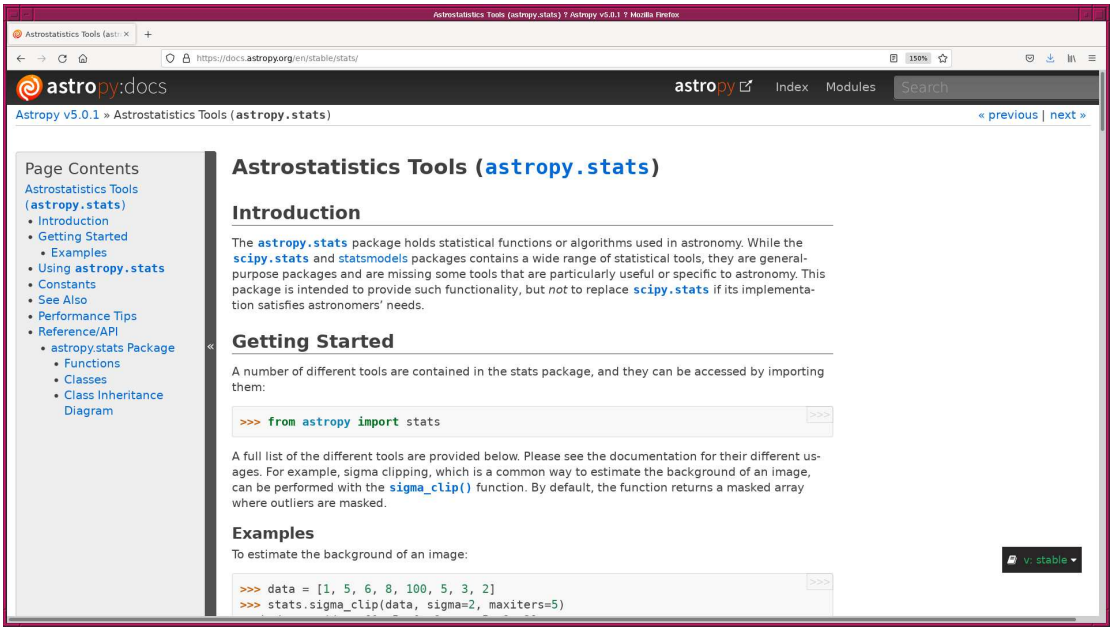

Figure 9: The official document of Astropy's astrostatistics sub-module.

12 A convenient way to give a rough estimate of readout noise

Here is a convenient way to give a crude estimate of readout noise. First, we choose two bias frames. Second, we subtract one bias frame from the other. Third, we calculate the standard deviation of difference of two bias frames. Then, a crude estimate of readout noise is given by dividing the standard deviation by $\sqrt{2}$.

Python Code 12: advobs2022021_s03_12.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib
```

```
# importing math module
import math

# importing numpy module
import numpy

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

# construction of parser object
desc = 'estimating readout noise from two bias frames'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-b1', '--bias1', default='bias1.fits', \
                     help='bias frame 1')
parser.add_argument ('-b2', '--bias2', default='bias2.fits', \
                     help='bias frame 2')
parser.add_argument ('-s', '--sigma', type=float, default=5.0, \
                     help='factor for sigma clipping')
parser.add_argument ('-n', '--nmaxiter', type=int, default=10, \
                     help='maximum number of iterations')

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

# input FITS file
file_bias1 = args.bias1
file_bias2 = args.bias2
nsigma     = args.sigma
nmaxiter   = args.nmaxiter

# if input file is not a FITS file, then skip
if not ( (file_bias1[-5:] == '.fits') and (file_bias2[-5:] == '.fits') ):
    # printing a message
    print ("Error: input file must be a FITS file!")
    # exit
    sys.exit ()

# file existence check using pathlib module
path_file_bias1 = pathlib.Path (file_bias1)
if not (path_file_bias1.exists ()):
    # printing a message
    print ("Error: input file \"%s\" does not exist!" % file_bias1)
    # exit
    sys.exit ()
path_file_bias2 = pathlib.Path (file_bias2)
if not (path_file_bias2.exists ()):
    # printing a message
    print ("Error: input file \"%s\" does not exist!" % file_bias2)
    # exit
    sys.exit ()

# file names
filename1 = path_file_bias1.name
filename2 = path_file_bias2.name
```

```
# opening FITS file
hdu_list1 = astropy.io.fits.open (file_bias1)
hdu_list2 = astropy.io.fits.open (file_bias2)

# primary HDU
hdu1 = hdu_list1[0]
hdu2 = hdu_list2[0]

# reading header
header1 = hdu1.header
header2 = hdu2.header

# reading data and conversion from uint16 into float64
bias1 = hdu1.data.astype (numpy.float64)
bias2 = hdu2.data.astype (numpy.float64)

# closing FITS file
hdu_list1.close ()
hdu_list2.close ()

# calculation of (bias1 - bias2)
diff_b1_b2 = bias1 - bias2

# sigma clipped mean and stddev
diff_sigclip  = astropy.stats.sigma_clip (diff_b1_b2, sigma=nsigma, \
                                          maxiters=nmaxiter, masked=False)
mean_sigclip   = numpy.mean (diff_sigclip)
stddev_sigclip = numpy.std  (diff_sigclip)
v_min_sigclip  = numpy.amin (diff_sigclip)
v_max_sigclip  = numpy.amax (diff_sigclip)

# printing result
print ("selected bias frames: %s and %s" % (filename1, filename2) )
print ("mean of bias1           = %f" % numpy.mean (bias1) )
print ("mean of bias2           = %f" % numpy.mean (bias2) )
print ("mean of diff            = %f" % numpy.mean (diff_b1_b2) )
print ("mean_sigclip            = %f" % mean_sigclip)
print ("stddev of diff          = %f ADU" % stddev_sigclip)
print ("estimated readout noise = %f ADU" % (stddev_sigclip / math.sqrt(2.0) ) )
```

Execute the script, and calculate readout noise.

```
% chmod a+x advobs202202_s03_12.py
% ./advobs202202_s03_12.py -b1 data_s03/lot_20210212_0079.fits \
? -b2 data_s03/lot_20210212_0080.fits
selected bias frames: lot_20210212_0079.fits and lot_20210212_0080.fits
mean of bias1           = 595.986696
mean of bias2           = 596.661863
mean of diff            = -0.675167
mean_sigclip            = -0.675066
stddev of diff          = 11.310364 ADU
estimated readout noise = 7.997635 ADU
```

Here is the other example.

```
% ./advobs202202_s03_12.py -b1 data_s03/lot_20210212_0100.fits \
? -b2 data_s03/lot_20210212_0101.fits
```

```
selected bias frames: lot_20210212_0100.fits and lot_20210212_0101.fits
mean of bias1           = 595.279329
mean of bias2           = 595.582915
mean of diff            = -0.303586
mean_sigclip            = -0.303573
stddev of diff          = 11.164789 ADU
estimated readout noise = 7.894698 ADU
```

13 Time variation of mean bias level

Make a Python script to check the time variation of mean bias level.

Python Code 13: advobs202202_s03_13.py

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

# 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
import astropy.stats

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

# construction of parser object
desc = 'Examining time variation of mean bias level'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('files', nargs='+', help='bias frames')
parser.add_argument ('-s', '--sigma', type=float, default=5.0, \
                     help='factor for sigma clipping (default: 5.0)')
parser.add_argument ('-n', '--nmaxiter', type=int, default=10, \
                     help='maximum number of iterations (default: 10)')
parser.add_argument ('-o', '--output', default='bias.png', \
                     help='output file name (default: bias.png)')
parser.add_argument ('-r', '--resolution', type=int, default=450, \
                     help='resolution of output image (default: 450)')

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

# input FITS file
list_files  = args.files
nsigma      = args.sigma
nmaxiter    = args.nmaxiter
```

```
file_output = args.output
resolution  = args.resolution

# data
data_datetime = numpy.array ([], dtype='datetime64[ms]')
data_mean     = numpy.array ([], dtype='float64')
data_stddev   = numpy.array ([], dtype='float64')

# if input file is not a FITS file, then skip
for file_fits in list_files:
    if not (file_fits[-5:] == '.fits'):
        # printing a message
        print ("Error: input file must be FITS files!")
        # exit
        sys.exit ()

# if output file is not either EPS, PDF, PNG, or PS, then skip
if not ( (file_output[-4:] == '.eps') or (file_output[-4:] == '.pdf') \
         or (file_output[-4:] == '.png') or (file_output[-3:] == '.ps') ):
    # printing a message
    print ("Error: output file must be a PNG or PDF or PS!")
    # exit
    sys.exit ()

# processing files
for file_fits in list_files:
    print ("Now processing the file \"%s\"..." % file_fits)

    # file existence check using pathlib module
    path_file_fits = pathlib.Path (file_fits)
    if not (path_file_fits.exists ()):
        # printing a message
        print ("Error: input file \"%s\" does not exist!" % file_fits)
        # skipping to next
        continue

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

    # primary HDU
    hdu0 = hdu_list[0]

    # reading header
    header0 = hdu0.header

    # reading data and conversion from uint16 into float64
    data0 = hdu0.data.astype (numpy.float64)

    # closing FITS file
    hdu_list.close ()

    # date/time
    date = header0['DATE-OBS']
    time = header0['TIME-OBS']
    datetime_str = "%sT%s" % (date, time)
    datetime64 = numpy.datetime64 (datetime_str)

    # sigma clipped mean and stddev
    data0_sigclip  = astropy.stats.sigma_clip (data0, sigma=nsigma, \
```

```
                                                    maxiters=nmaxiter, masked=False)
    mean_sigclip   = numpy.mean (data0_sigclip)
    stddev_sigclip = numpy.std  (data0_sigclip)

    # appending data to the lists
    data_datetime = numpy.append (data_datetime, datetime64)
    data_mean     = numpy.append (data_mean, mean_sigclip)
    data_stddev   = numpy.append (data_stddev, stddev_sigclip)

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

# labels
label_x = 'Date/Time [UT]'
label_y = 'Mean Bias Level [ADU]'
ax.set_xlabel (label_x)
ax.set_ylabel (label_y)
ax.xaxis.set_major_formatter (matplotlib.dates.DateFormatter ('%H:%M'))

# axis settings
y_min = 580.0
y_max = 610.0
ax.set_ylim (y_min, y_max)

# plotting data
ax.errorbar (data_datetime, data_mean, yerr=data_stddev, \
             marker='o', color='blue', markersize=3, linestyle='none', \
             ecolor='black', capsize=2, \
             label='Mean bias level')
ax.legend ()

# saving the figure to a file
fig.savefig (file_output, dpi=resolution)
```

Run the script, and generate a PNG file.

```
% chmod a+x advobs202202_s03_13.py
% ./advobs202202_s03_13.py -h
usage: advobs202202_s03_13.py [-h] [-s SIGMA] [-n NMAXITER] [-o OUTPUT]
                              [-r RESOLUTION]
                              files [files ...]

Examining time variation of mean bias level

positional arguments:
  files                 bias frames

optional arguments:
  -h, --help            show this help message and exit
  -s SIGMA, --sigma SIGMA
                        factor for sigma clipping (default: 5.0)
  -n NMAXITER, --nmaxiter NMAXITER
                        maximum number of iterations (default: 10)
  -o OUTPUT, --output OUTPUT
                        output file name (default: bias.png)
  -r RESOLUTION, --resolution RESOLUTION
```

```
                        resolution of output image (default: 450)

% ./advobs202202_s03_13.py -o biasleve.png data_s03/*.fits
Now processing the file "data_s03/lot_20210212_0079.fits"...
Now processing the file "data_s03/lot_20210212_0080.fits"...
Now processing the file "data_s03/lot_20210212_0081.fits"...
Now processing the file "data_s03/lot_20210212_0082.fits"...
Now processing the file "data_s03/lot_20210212_0083.fits"...
Now processing the file "data_s03/lot_20210212_0084.fits"...
Now processing the file "data_s03/lot_20210212_0100.fits"...
Now processing the file "data_s03/lot_20210212_0101.fits"...
Now processing the file "data_s03/lot_20210212_0102.fits"...
Now processing the file "data_s03/lot_20210212_0112.fits"...

.....

Now processing the file "data_s03/lot_20210212_0809.fits"...
Now processing the file "data_s03/lot_20210212_0810.fits"...
Now processing the file "data_s03/lot_20210212_0811.fits"...
Now processing the file "data_s03/lot_20210212_0812.fits"...
Now processing the file "data_s03/lot_20210212_0813.fits"...
Now processing the file "data_s03/lot_20210212_0814.fits"...
Now processing the file "data_s03/lot_20210212_0815.fits"...
Now processing the file "data_s03/lot_20210212_0816.fits"...
Now processing the file "data_s03/lot_20210212_0817.fits"...
Now processing the file "data_s03/lot_20210212_0818.fits"...
% ls -l biasleve.png
-rw-r--r--  1 daisuke  taiwan  149321 Mar  4 01:18 biasleve.png
```

Show the PNG image using an image viewer program. (Fig. 10)

```
% feh -dF biaslevel.png
```

Roughly speaking, the bias level seems to be stable over time.

14 For your further reading

- Read chapter 4 of "Handbook of CCD Astronomy" and learn about bias frame.
  - Handbook of CCD Astronomy (2nd Edition)
    - Steve B. Howell
    - Cambridge University Press
    - `https://doi.org/10.1017/CBO9780511807909`
- Visit the official website of Numpy, and learn about Numpy arrays.
  - `https://numpy.org/`
- Visit the official website of Matplotlib, and learn about usage of Matplotlib.
  - `https://matplotlib.org/`
- Visit the official website of Astropy, and learn about sigma clipping algorithm.
  - `https://docs.astropy.org/en/stable/stats/index.html`
  - `https://docs.astropy.org/en/stable/stats/robust.html`

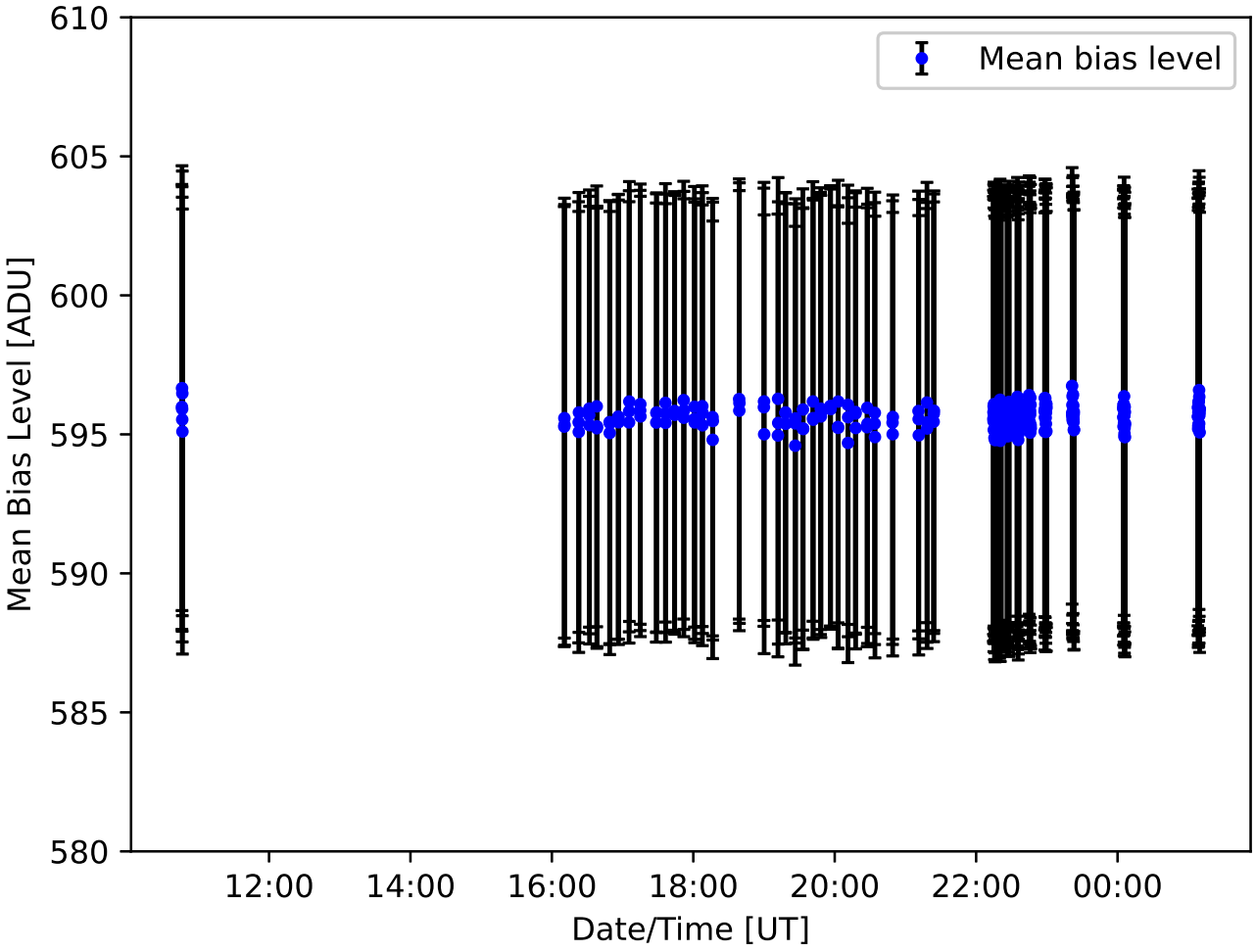

Figure 10: Time variation of mean bias level.

15 Exercise

1. What is bias? Describe bias. How to take bias frames? Why do we need to take bias frames during the observing run? Why mean value of bias is not zero? How useful and important are bias frames?
2. Choose three bias frames. Make your own Python script to visualise those three bias frames using Matplotlib. Show the source code of your Python script and images you have produced.
3. Choose three bias frames. Make your own Python script to make histograms of those three bias frames using Matplotlib. Show the source code of your Python script and histograms you have produced. Explain how you decide the width of the bin for histogram.
4. Choose three bias frames. Make your own Python script to calculate mean, median, variance, standard deviation, minimum value, maximum value of those three bias frames without using convenient functions of Numpy (e.g. numpy.mean, numpy.std, etc.). Show the source code of your Python script. Take a screenshot of your computer display after the execution of your Python script, and show the result of your calculation.
5. Choose three bias frames. Make your own Python script to calculate mean, median, variance, standard deviation, minimum value, maximum value of those three bias frames using convenient functions of Numpy (e.g. numpy.mean, numpy.std, etc.). Show the source code of your Python script. Take a screenshot of your computer display after the execution of your Python script, and show the result of your calculation.
6. Choose one bias frame. Set four regions of 512×512 pixels on the image. Make your own Python script to calculate mean, median, variance, standard deviation, minimum value, maximum value of those four regions. Discuss the uniformity of the bias frame.
7. What is sigma clipping algorithm? How useful is it? Give some examples that we should use sigma clipping. Describe why sigma clipping is more robust.
8. Other than sigma clipping algorithm, is there any other robust estimator? If so, explain the method.
9. Make your own function which works like `astropy.stats.sigma_clip ()`. Show the source code of your function. Test your function with some bias data for this session. Take a screenshot of your computer display to show the results.
10. Make your own Python script which works like `imstatistics` task of IRAF (Image Reduction and Analysis Facility). Implement sigma clipping for your Python script. Show the source code of your function. Test your script with some bias data for this session. Take a screenshot of your computer display to show the results.
    - `https://iraf.net/irafhelp.php?val=imstatistics&help=Help+Page`
11. To estimate readout noise, we first subtracted one bias frame from another. Describe why we did this. Then, we estimated standard deviation of difference of two bias frames and divided standard deviation by $\sqrt{2}$. Describe why we divide the standard deviation by $\sqrt{2}$.
12. Choose two bias frames. Make a Python script to subtract one bias from another. Visualise difference of two bias frames using Matplotlib. Show the image you have produced.
13. Make your own Python script to show time variation of mean bias level. Use Matplotlib to produce a plot. Show the source code of your Python script and plot you have produced. Discuss the time variability of bias frames.