Advanced Astronomical Observations 2022
Session 10: Centroid and PSF Fitting

Kinoshita Daisuke

08 April 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/`

We often measure a position of an astronomical object on image. For this session, we try centroid measurements and PSF (Point-Spread Function) fitting of stellar images.

1 Photutils package

For this session, `photutils` package is needed. Visit following websites to learn about `photutils`. (Fig. 1) The `photutils` package offers many useful functions for astronomical photometry. (2)

- photutils
  - `https://photutils.readthedocs.io/`
  - `https://pypi.org/project/photutils/`
  - `https://github.com/astropy/photutils`

Try following to check whether you have `photutils` package installed on your computer. If you can import `photutils` package successfully, then you have `photutils` package properly installed on your computer.

```
% python3.9
Python 3.9.12 (main, Apr  4 2022, 09:16:48)
[GCC 10.3.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import photutils
>>> exit ()
```

If you see an error message like below, then you do not have `photutils` package on your computer.

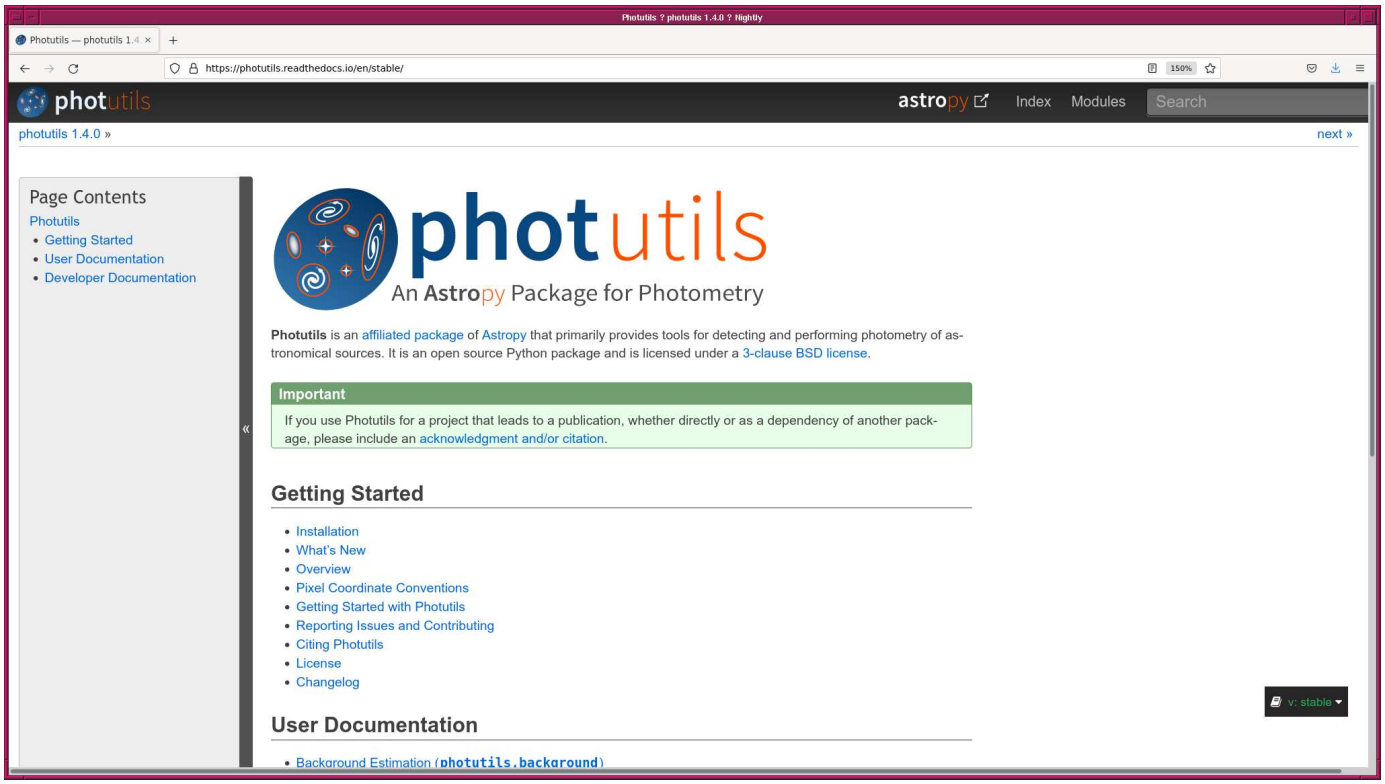

Figure 1: The official website of `photutils` package at `https://photutils.readthedocs.io/`.

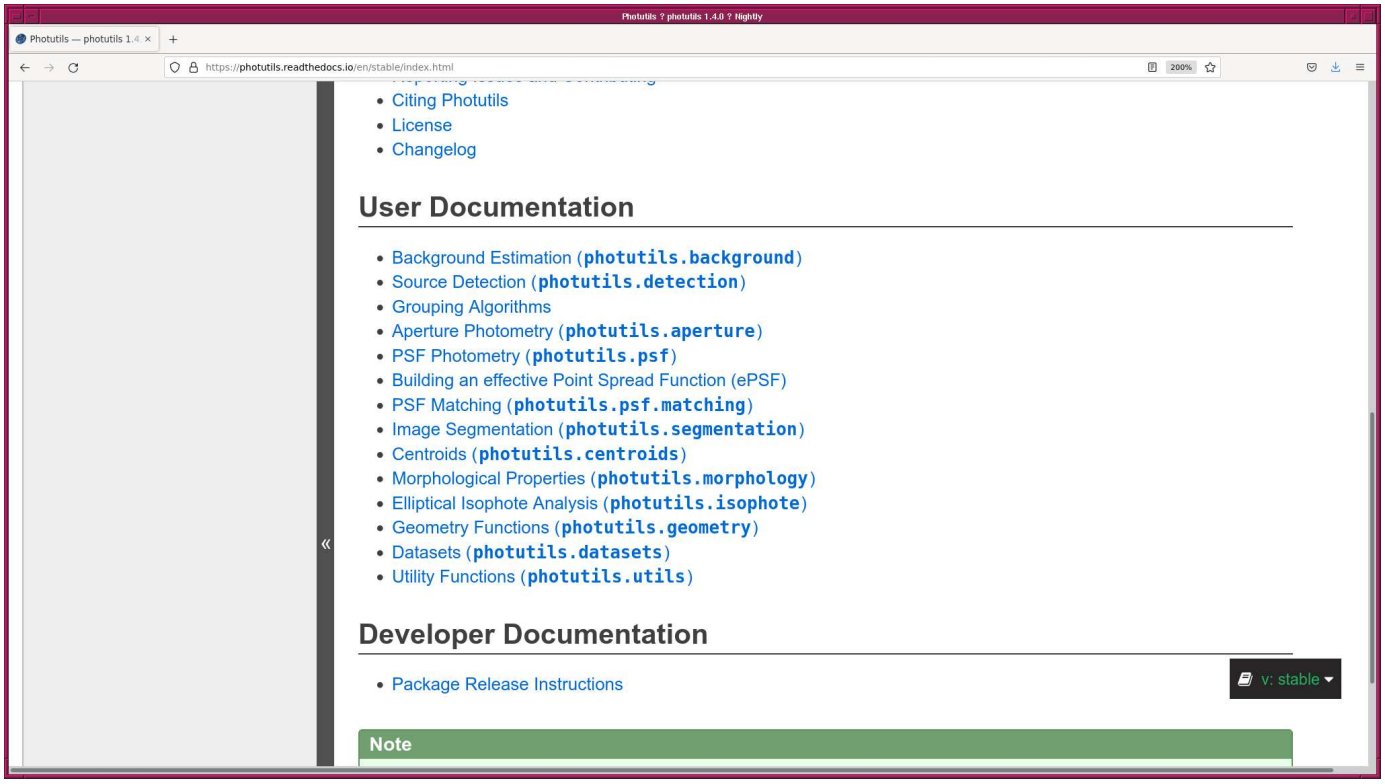

Figure 2: The user documentation of `photutils` on the official website of `photutils` package at `https://photutils.readthedocs.io/`.

```
% python3.9
Python 3.9.12 (main, Apr  6 2022, 23:48:29)
[GCC 10.3.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import photutils
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'photutils'
>>> exit ()
```

If you do not have `photutils` pakcage, visit the official website of `photutils` package, read the installation instruction (Fig. 3), and install the package on your computer.

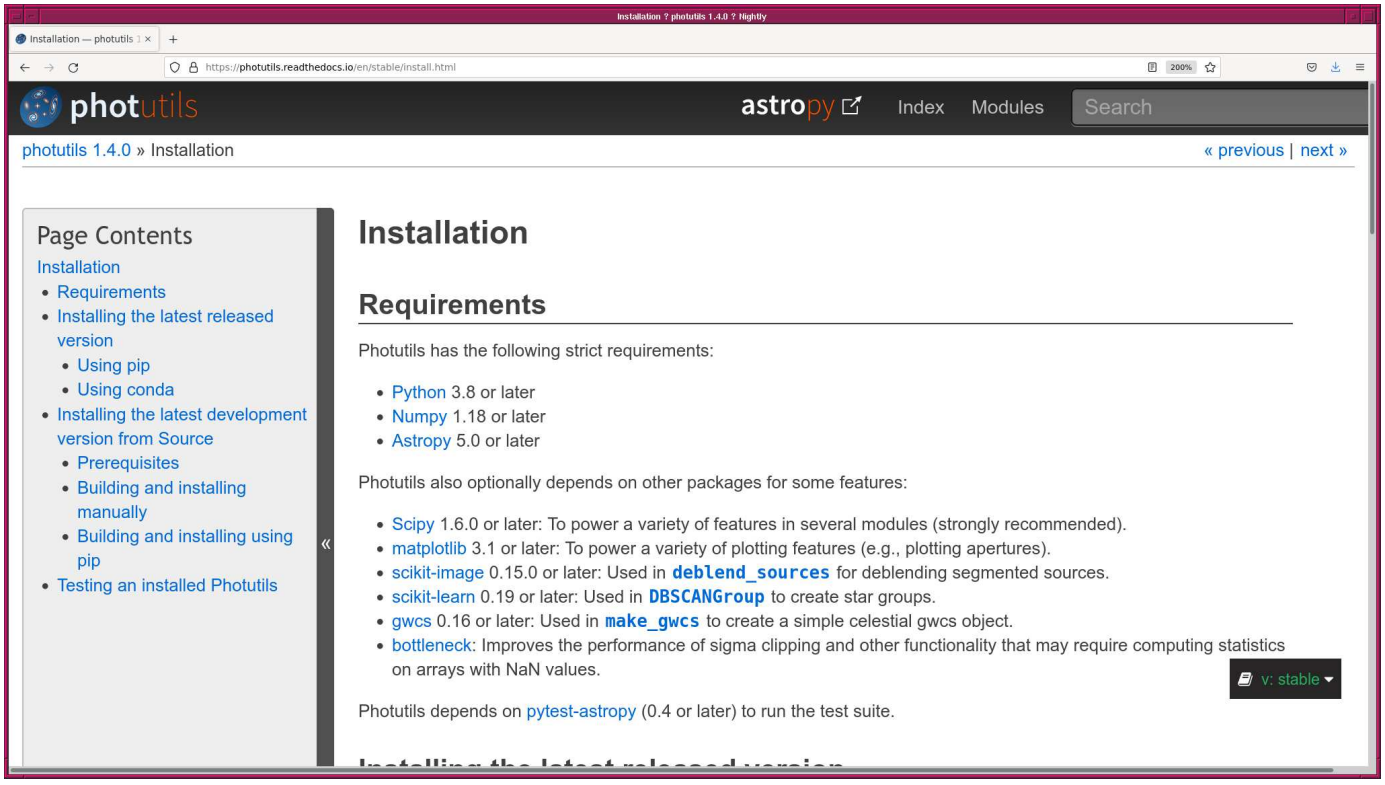

Figure 3: The installation instruction and description of the requirements at the official website of `photutils` package at `https://photutils.readthedocs.io/en/stable/install.html`.

2 Generating synthetic images

Use `photutils.datasets` to generate synthetic images.

2.1 Making a background image

Make a Python script to generate an image simulating sky background.

Python Code 1: advobs202202_s10_01.py

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

#
# Time-stamp: <2022/04/07 13:34:46 (CST) daisuke>
```

```
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing datetime module
import datetime

# importing astropy module
import astropy.io.fits

# importing photutils module
import photutils.datasets

# constructing parser object
desc = 'generating a synthetic image'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-b', '--background', type=float, default=1000.0, \
                     help='background level (default: 1000)')
parser.add_argument ('-s', '--sigma', type=float, default=10.0, \
                     help='noise level (default: 10)')
parser.add_argument ('-x', '--xsize', type=int, default=512, \
                     help='image size in x-axis (default: 512 pixel)')
parser.add_argument ('-y', '--ysize', type=int, default=512, \
                     help='image size in y-axis (default: 512 pixel)')
parser.add_argument ('-o', '--output', default='', \
                     help='output file name')

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

# input parameters
sky_background_level = args.background
noise_level          = args.sigma
image_size_x         = args.xsize
image_size_y         = args.ysize
file_output          = args.output

# making pathlib object
path_output = pathlib.Path (file_output)

# checking output file name
if (file_output == ''):
    # printing message
    print ("You need to specify output file name.")
    # exit
    sys.exit ()
# output file must be a FITS file
if not (path_output.suffix == '.fits'):
    # printing message
    print ("Output file must be a FITS file.")
```

```
    # exit
    sys.exit ()
# existence check of output file
if (path_output.exists ()):
    # printing message
    print ("Output file exists. Exiting...")
    # exit
    sys.exit ()

# image size
image_size = (image_size_x, image_size_y)

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

# command name
command = sys.argv[0]

# printing message
print ("#")
print ("# input parameters")
print ("#")
print ("#  image size                = %d x %d" % (image_size_x, image_size_y) )
print ("#  mean sky background level = %f ADU" % sky_background_level)
print ("#  noise level (stddev)      = %f ADU" % noise_level)
print ("#")

# printing status
print ("# now, generating image...")

# generating sky background
image_background \
    = photutils.datasets.make_noise_image (image_size, \
                                           distribution='gaussian', \
                                           mean=sky_background_level, \
                                           stddev=noise_level)

# printing status
print ("# finished generating image!")

# printing status
print ("# now, generating FITS header...")

# preparing a FITS header
header = astropy.io.fits.PrimaryHDU ().header

# adding comments to the header
header['history'] = "FITS file created by the command \"%s\"" % (command)
header['history'] = "Updated on %s" % (now)
header['comment'] = "synthetic astronomical image simulating sky background"
header['comment'] = "Options given:"
header['comment'] = "  image size = %d x %d" % (image_size_x, image_size_y)
header['comment'] = "  mean sky background level = %f ADU" \
    % (sky_background_level)
header['comment'] = "  noise level  = %f ADU" % (noise_level)

# printing status
print ("# finished generating FITS header!")
```

```
# printing status
print ("# now, writing output FITS file...")

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

# printing status
print ("# finished writing output FITS file!")
```

Execute the script and generate a synthetic image.

```
% chmod a+x advobs202202_s10_01.py
% ./advobs202202_s10_01.py -h
usage: advobs202202_s10_01.py [-h] [-b BACKGROUND] [-s SIGMA] [-x XSIZE]
                              [-y YSIZE] [-o OUTPUT]

generating a synthetic image

optional arguments:
  -h, --help            show this help message and exit
  -b BACKGROUND, --background BACKGROUND
                        background level (default: 1000)
  -s SIGMA, --sigma SIGMA
                        noise level (default: 10)
  -x XSIZE, --xsize XSIZE
                        image size in x-axis (default: 512 pixel)
  -y YSIZE, --ysize YSIZE
                        image size in y-axis (default: 512 pixel)
  -o OUTPUT, --output OUTPUT
                        output file name

% ./advobs202202_s10_01.py -b 2000 -s 30 -x 1024 -y 1024 -o synthetic_00.fits
#
# input parameters
#
#  image size                = 1024 x 1024
#  mean sky background level = 2000.000000 ADU
#  noise level (stddev)      = 30.000000 ADU
#
# now, generating image...
# finished generating image!
# now, generating FITS header...
# finished generating FITS header!
# now, writing output FITS file...
# finished writing output FITS file!
% ls -lF *.fits
-rw-r--r--  1 daisuke  taiwan  8392320 Apr  7 13:34 synthetic_00.fits
% file *.fits
synthetic_00.fits: FITS image data, 64-bit, floating point, double precision
```

2.2 Examining pixel values

Make a Python script to examine pixel values of generated synthetic image.

Python Code 2: advobs202202_s10_02.py

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

```
#
# Time-stamp: <2022/04/07 13:46:04 (CST) daisuke>
#

# importing argparse module
import argparse

# importing pathlib module
import pathlib

# importing numpy module
import numpy
import numpy.ma

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

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

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

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

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

# printing information
print ("# Input parameters")
print ("#   rejection algorithm = %s" % rejection)
if not (rejection == 'NONE'):
    print ("#   threshold of sigma-clipping = %f" % threshold)
    print ("#   maximum number of iterations = %d" % maxiters)

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

# scanning files
for file_fits in list_files:
    # making pathlib object
    path_fits = pathlib.Path (file_fits)
```

```
    # if the file is not a FITS file, then skip
    if not (path_fits.suffix == '.fits'):
        # printing message
        print ("# skipping file '%s'..." % file_fits)
        # skipping
        continue

    # opening FITS file
    with astropy.io.fits.open (file_fits) as hdu_list:
        # reading header of primary HDU
        header = hdu_list[0].header
        # reading image of primary HDU
        data0 = hdu_list[0].data

    # calculations

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

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

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

    # file name
    filename = path_fits.name
```

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

Run the script and check pixel values.

```
% chmod a+x advobs202202_s10_02.py
% ./advobs202202_s10_02.py -h
usage: advobs202202_s10_02.py [-h] [-r {NONE,sigclip}] [-t THRESHOLD]
                              [-n MAXITERS]
                              files [files ...]

calculating statistical information of FITS files

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -r {NONE,sigclip}, --rejection {NONE,sigclip}
                        rejection algorithm (default: NONE)
  -t THRESHOLD, --threshold THRESHOLD
                        threshold for sigma clipping (default: 4.0)
  -n MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 10)

% ./advobs202202_s10_02.py synthetic_00.fits
# Input parameters
#   rejection algorithm = NONE
#
# file name                    npix     mean   median   stddev      min      max
#
synthetic_00.fits            1048576  2000.00  1999.98   29.99  1862.73  2147.08
```

The mean and stddev are ~ 2000 and ~ 30, respectively, as expected.

2.3 Visual inspection of the image

Use Ginga to show the image. (Fig. 4)

```
% ginga synthetic_00.fits
```

2.4 Generating a synthetic image with stars

Make a Python script to generate a synthetic image simulating sky background and stars.

Python Code 3: advobs202202_s10_03.py

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

#
# Time-stamp: <2022/04/07 14:22:41 (CST) daisuke>
#

# importing argparse module
import argparse
```

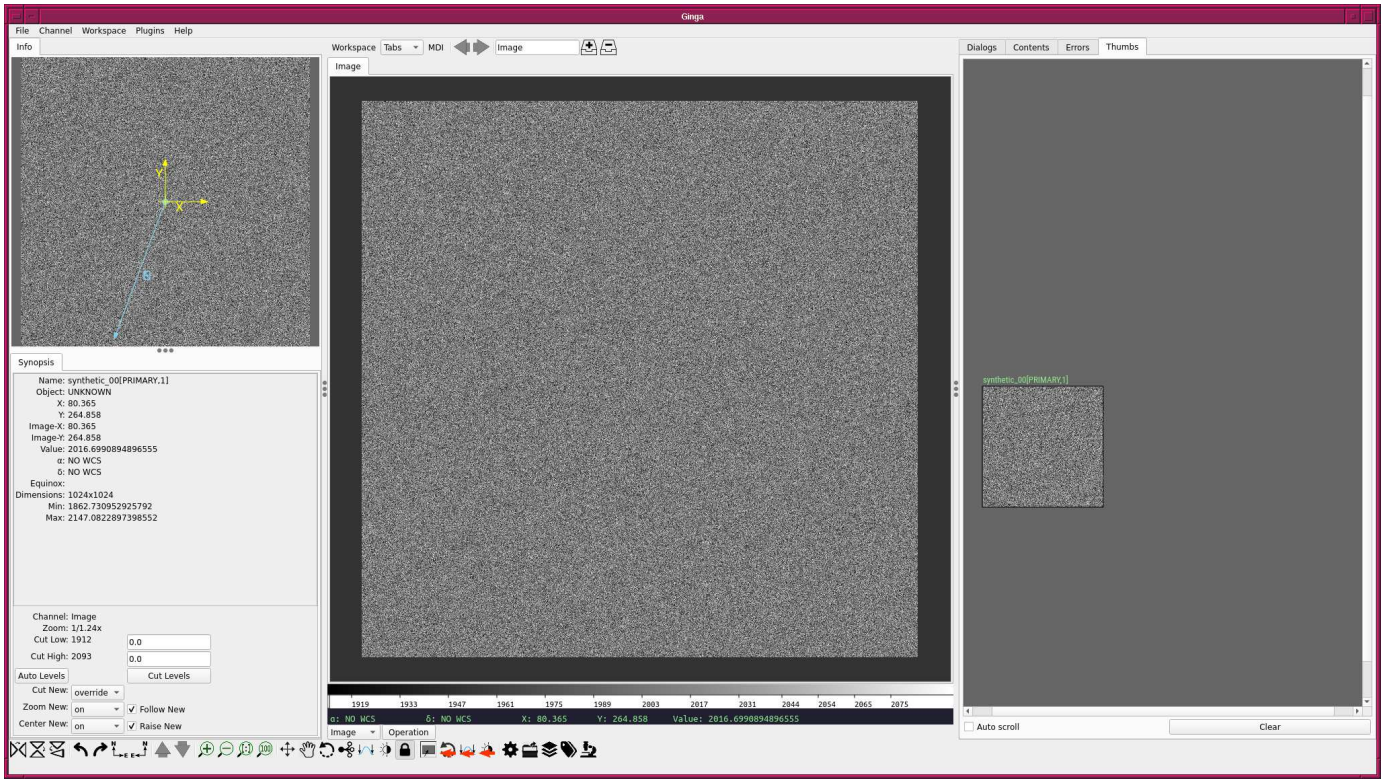

Figure 4: The synthetic image generated by photutils.datasets module.

```
# importing sys module
import sys

# importing pathlib module
import pathlib

# importing datetime module
import datetime

# importing numpy module
import numpy

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

# importing photutils module
import photutils.datasets

# constructing parser object
desc   = 'generating a synthetic image'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-b', '--background', type=float, default=1000.0, \
                     help='background level (default: 1000)')
parser.add_argument ('-s', '--sigma', type=float, default=10.0, \
                     help='noise level (default: 10)')
parser.add_argument ('-n', '--nstar', type=int, default=10, \
```

```
                      help='number of stars to add (default: 10)')
parser.add_argument ('-f', '--flux', type=float, default=10000.0, \
                      help='maximum total flux of star (default: 10000)')
parser.add_argument ('-p', '--psf', type=float, default=5.0, \
                      help='FWHM of stellar radial profile (default: 5)')
parser.add_argument ('-x', '--xsize', type=int, default=512, \
                      help='image size in x-axis (default: 512)')
parser.add_argument ('-y', '--ysize', type=int, default=512, \
                      help='image size in y-axis (default: 512)')
parser.add_argument ('-o', '--output', default='', \
                      help='output file name')

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

# input parameters
sky_background_level = args.background
noise_level          = args.sigma
nstar                = args.nstar
flux_max             = args.flux
psf_fwhm             = args.psf
image_size_x         = args.xsize
image_size_y         = args.ysize
file_output          = args.output

# making pathlib object
path_output = pathlib.Path (file_output)

# checking output file name
if (file_output == ''):
    # printing message
    print ("You need to specify output file name.")
    # exit
    sys.exit ()
# output file must be a FITS file
if not (path_output.suffix == '.fits'):
    # printing message
    print ("Output file must be a FITS file.")
    # exit
    sys.exit ()
# existence check of output file
if (path_output.exists ()):
    # printing message
    print ("Output file exists. Exiting...")
    # exit
    sys.exit ()

# image size
image_size = (image_size_x, image_size_y)

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

# command name
command = sys.argv[0]

# printing message
print ("#")
print ("# input parameters")
```

```
print ("#")
print ("#  image size                = %d x %d" \
       % (image_size[0], image_size[1]) )
print ("#  mean sky background level = %f ADU" % sky_background_level)
print ("#  noise level (stddev)      = %f ADU" % noise_level)
print ("#  number of stars to add    = %d" % nstar)
print ("#  FWHM of stellar PSF       = %f pixel" % psf_fwhm)
print ("#  max total flux of stars   = %f ADU" % flux_max)
print ("#")

# printing status
print ("# now, generating background image...")

# generating sky background
image_background \
    = photutils.datasets.make_noise_image (image_size, \
                                           distribution='gaussian', \
                                           mean=sky_background_level, \
                                           stddev=noise_level)

# printing status
print ("# finished generating background image!")

# printing status
print ("# now, generating source table...")

# making a source table
fwhm_sigma            = 2.0 * numpy.sqrt (2.0 * numpy.log (2.0) )
source_table          = astropy.table.Table ()
source_table['x_0']   = numpy.random.normal (image_size_x * 0.5, \
                                             image_size_x * 0.1, nstar)
source_table['y_0']   = numpy.random.normal (image_size_y * 0.5, \
                                             image_size_y * 0.1, nstar)
source_table['sigma'] = numpy.array ([psf_fwhm] * nstar) / fwhm_sigma
source_table['flux']  = numpy.random.uniform (flux_max * 0.1, flux_max, nstar)

# printing status
print ("# finished generating source table!")

# printing source table
print ("# source_table:")
for i in range (nstar):
    print ("#   a star of flux %8.1f ADU at (x,y)=(%7.1f,%7.1f)" \
           % (source_table['flux'][i], source_table['x_0'][i], \
              source_table['y_0'][i]) )
print ("# total number of stars in source table = %d" % len (source_table))

# printing status
print ("# now, generating stars...")

# generating stars
image_star \
    = photutils.datasets.make_gaussian_prf_sources_image (image_size, \
                                                          source_table)

# printing status
print ("# finished generating stars!")

# printing status
```

```
print ("# now generating stars + background image...")

# making synthetic image
image = image_background + image_star

# printing status
print ("# now, generating FITS header...")

# preparing a FITS header
header = astropy.io.fits.PrimaryHDU ().header

# adding comments to the header
header['history'] = "FITS file created by the command \"%s\"" % (command)
header['history'] = "Updated on %s" % (now)
header['comment'] = "synthetic astronomical image simulating sky background"
header['comment'] = "Options given:"
header['comment'] = "  image size = %d x %d" % (image_size_x, image_size_y)
header['comment'] = "  mean sky background level = %f ADU" \
    % (sky_background_level)
header['comment'] = "  noise level  = %f ADU" % (noise_level)
header['comment'] = "  number of stars  = %d" % (nstar)
header['comment'] = "  FWHM of stellar PSF = %f pix" % (psf_fwhm)
header['comment'] = "  max total flux of stars  = %f ADU" % (flux_max)

# printing status
print ("# finished generating FITS header!")

# printing status
print ("# finished generating stars + background image!")

# printing status
print ("# now, writing output FITS file...")

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

# printing status
print ("# finished writing output FITS file!")
```

Run the script and generate a synthetic image with 500 stars.

```
% chmod a+x advobs202202_s10_03.py
% ./advobs202202_s10_03.py -h
usage: advobs202202_s10_03.py [-h] [-b BACKGROUND] [-s SIGMA] [-n NSTAR]
                              [-f FLUX] [-p PSF] [-x XSIZE] [-y YSIZE]
                              [-o OUTPUT]

generating a synthetic image

optional arguments:
  -h, --help            show this help message and exit
  -b BACKGROUND, --background BACKGROUND
                        background level (default: 1000)
  -s SIGMA, --sigma SIGMA
                        noise level (default: 10)
  -n NSTAR, --nstar NSTAR
                        number of stars to add (default: 10)
  -f FLUX, --flux FLUX  maximum total flux of star (default: 10000)
```

```
  -p PSF, --psf PSF     FWHM of stellar radial profile (default: 5)
  -x XSIZE, --xsize XSIZE
                        image size in x-axis (default: 512)
  -y YSIZE, --ysize YSIZE
                        image size in y-axis (default: 512)
  -o OUTPUT, --output OUTPUT
                        output file name

% ./advobs202202_s10_03.py -b 2000 -s 30 -n 500 -f 50000 -p 5 -x 1024 -y 1024 \
? -o synthetic_01.fits
#
# input parameters
#
#  image size                = 1024 x 1024
#  mean sky background level = 2000.000000 ADU
#  noise level (stddev)      = 30.000000 ADU
#  number of stars to add    = 500
#  FWHM of stellar PSF       = 5.000000 pixel
#  max total flux of stars   = 50000.000000 ADU
#
# now, generating background image...
# finished generating background image!
# now, generating source table...
# finished generating source table!
# source_table:
#   a star of flux  45511.9 ADU at (x,y)=(  556.1,  420.0)
#   a star of flux  29539.5 ADU at (x,y)=(  464.0,  423.1)
#   a star of flux  25761.1 ADU at (x,y)=(  714.1,  573.1)
#   a star of flux  24531.3 ADU at (x,y)=(  409.6,  505.1)
#   a star of flux  24741.1 ADU at (x,y)=(  609.1,  355.4)
#   a star of flux  29594.8 ADU at (x,y)=(  635.3,  556.2)
#   a star of flux  11753.5 ADU at (x,y)=(  453.2,  586.7)
#   a star of flux  23622.1 ADU at (x,y)=(  519.7,  455.1)
#   a star of flux  41629.9 ADU at (x,y)=(  527.1,  301.6)
#   a star of flux  17926.6 ADU at (x,y)=(  542.1,  487.3)

.....

#   a star of flux   7068.0 ADU at (x,y)=(  344.8,  615.3)
#   a star of flux  19452.7 ADU at (x,y)=(  502.6,  416.7)
#   a star of flux  16209.5 ADU at (x,y)=(  614.6,  582.0)
#   a star of flux  31717.9 ADU at (x,y)=(  684.5,  446.1)
#   a star of flux   7260.5 ADU at (x,y)=(  415.8,  529.5)
#   a star of flux  18148.8 ADU at (x,y)=(  560.4,  426.0)
#   a star of flux  45924.5 ADU at (x,y)=(  465.7,  549.3)
#   a star of flux  49040.1 ADU at (x,y)=(  628.2,  614.0)
#   a star of flux  41541.0 ADU at (x,y)=(  489.2,  571.0)
#   a star of flux  13223.4 ADU at (x,y)=(  516.4,  328.6)
# total number of stars in source table = 500
# now, generating stars...
# finished generating stars!
# now generating stars + background image...
# now, generating FITS header...
# finished generating FITS header!
# finished generating stars + background image!
# now, writing output FITS file...
# finished writing output FITS file!
% ls -lF *.fits
-rw-r--r--  1 daisuke  taiwan  8392320 Apr  7 13:34 synthetic_00.fits
```

```
-rw-r--r--  1 daisuke  taiwan  8392320 Apr  7 14:23 synthetic_01.fits
% file synthetic_0*
synthetic_00.fits: FITS image data, 64-bit, floating point, double precision
synthetic_01.fits: FITS image data, 64-bit, floating point, double precision
```

2.5 Visual inspection of the image

Use Ginga to show the image. (Fig. 5)

```
% ginga synthetic_01.fits
```

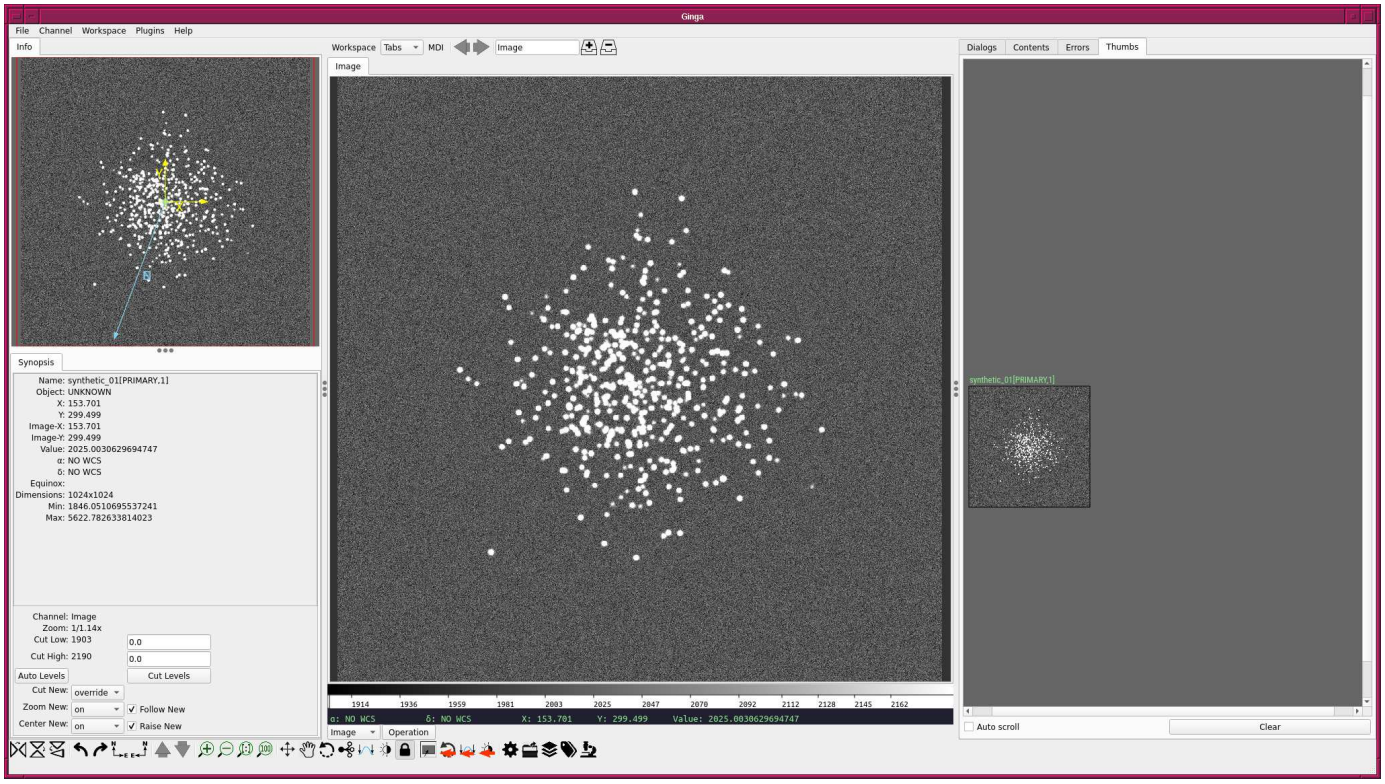

Figure 5: The synthetic image with stars generated by photutils.datasets module.

3 Centroid measurements

To know the location of a star, centroid is often used.

3.1 Knowing a rough position of a star using Ginga

Before starting centroid measurement, use Ginga to know a rough position of a star.

1. Start Ginga. (Fig. 6)
2. Load the image "`synthetic_01.fits`". (Fig. 7 and 8)
3. Pick a star for your centroid measurement. Choose a relatively isolated star.
4. Zoom in and pan to the star you have picked. (Fig. 9)
5. Move your mouse cursor to your target.
6. Read rough values of (x, y) coordinate of your target. (Fig. 10)

For example, a star at (x, y) $\sim$ (412, 277) is selected.

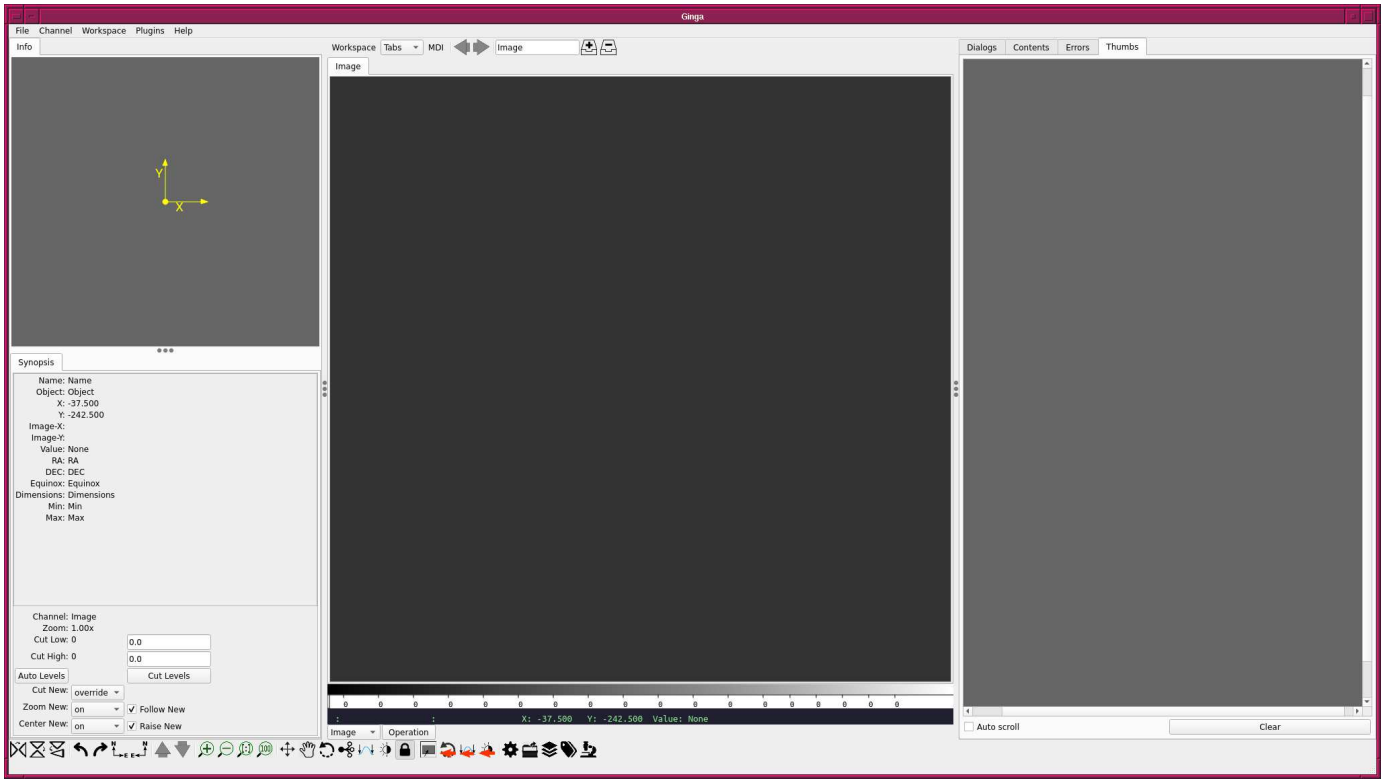

Figure 6: Ginga just after the start-up.

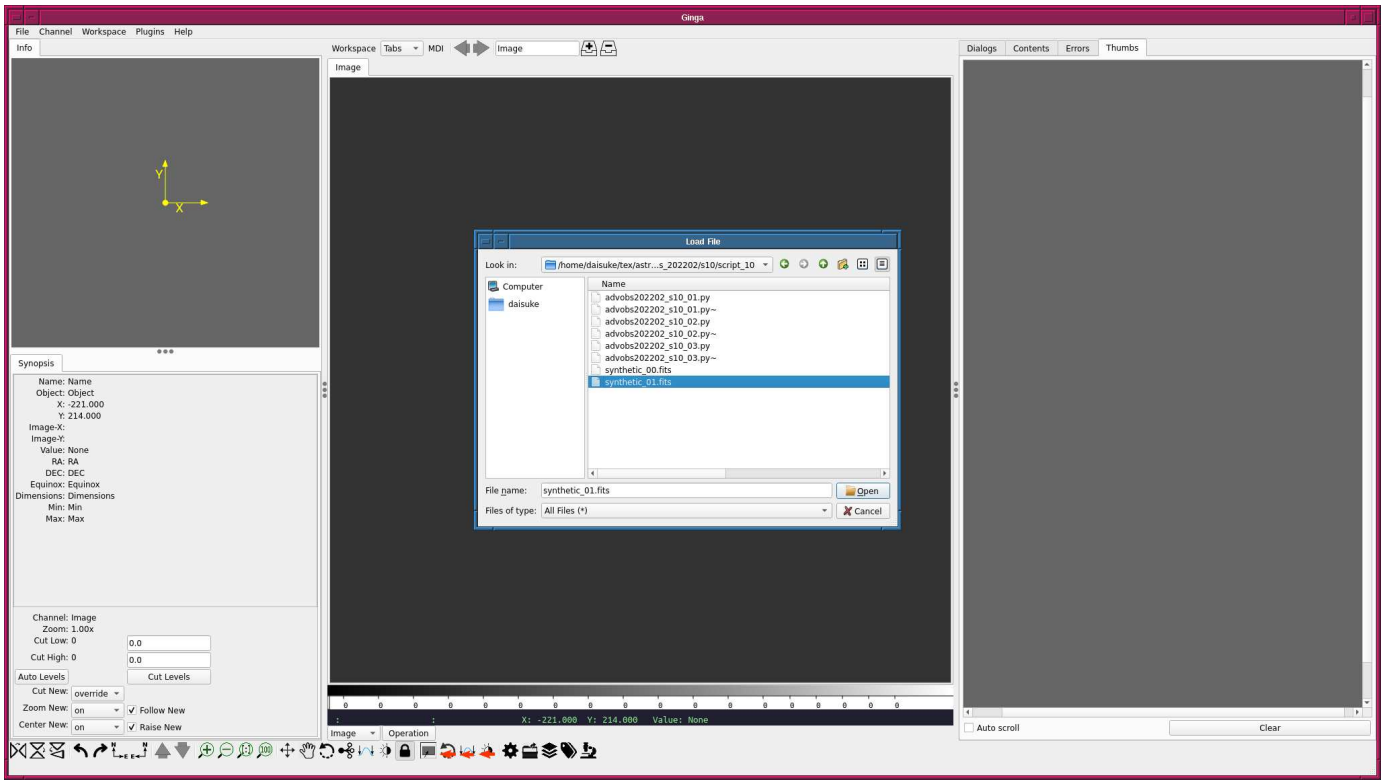

Figure 7: Opening a FITS file using Ginga.

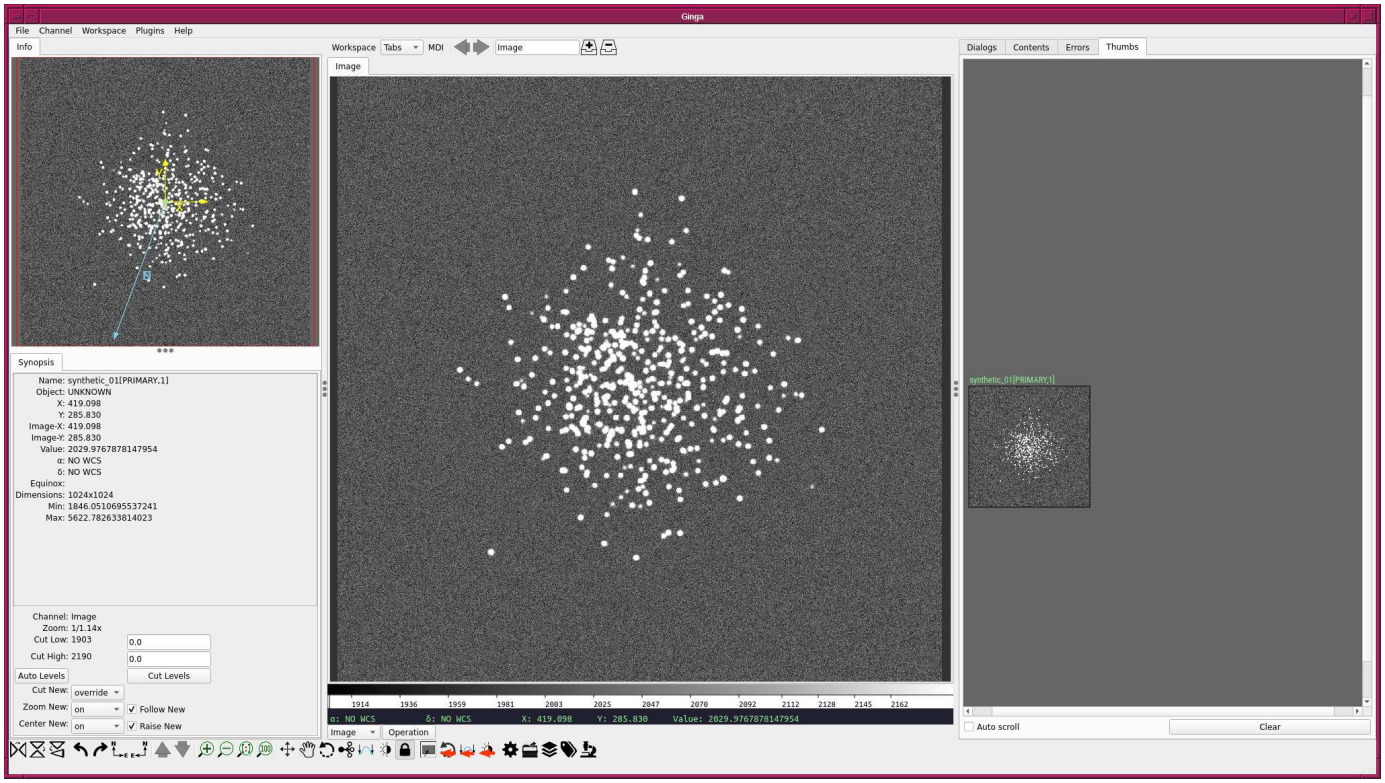

Figure 8: A FITS image is opened by Ginga.

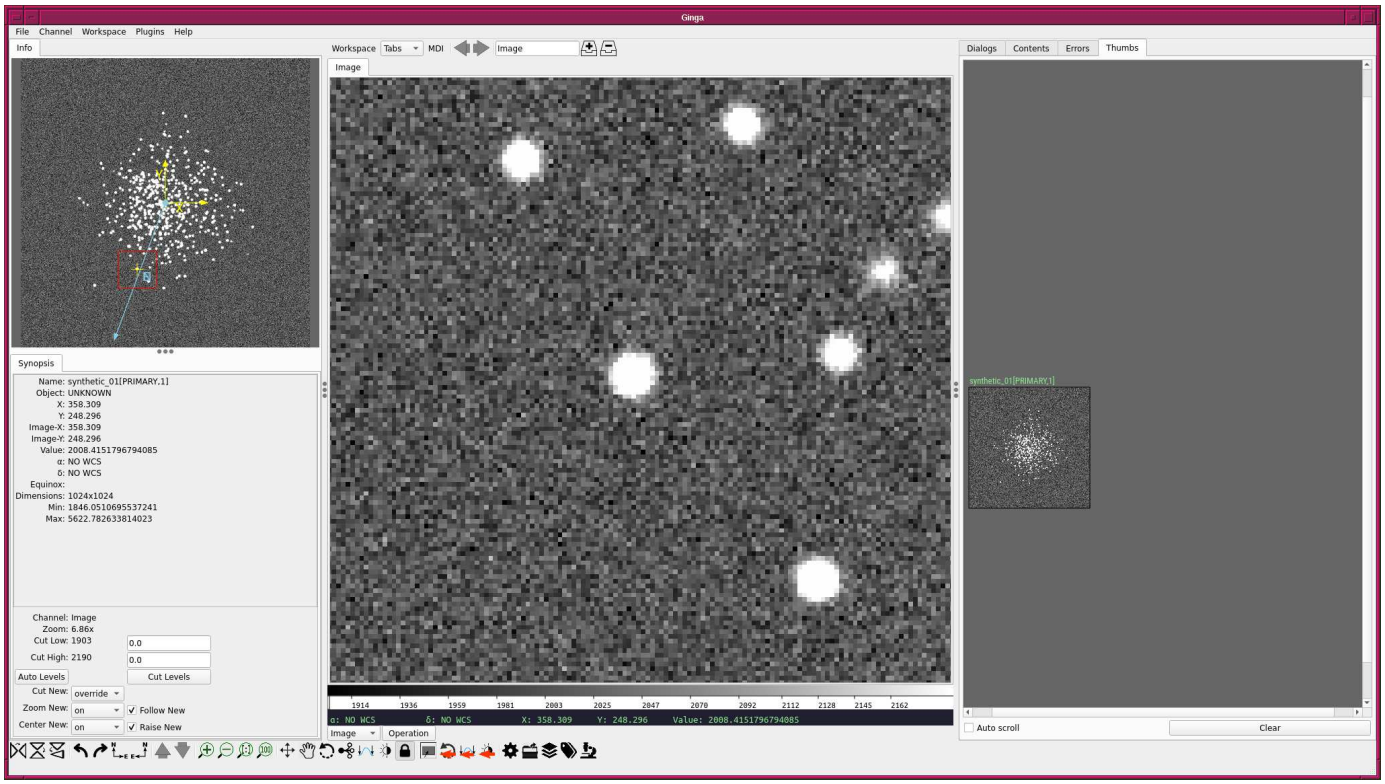

Figure 9: Selecting a star and zooming and panning to selected star.

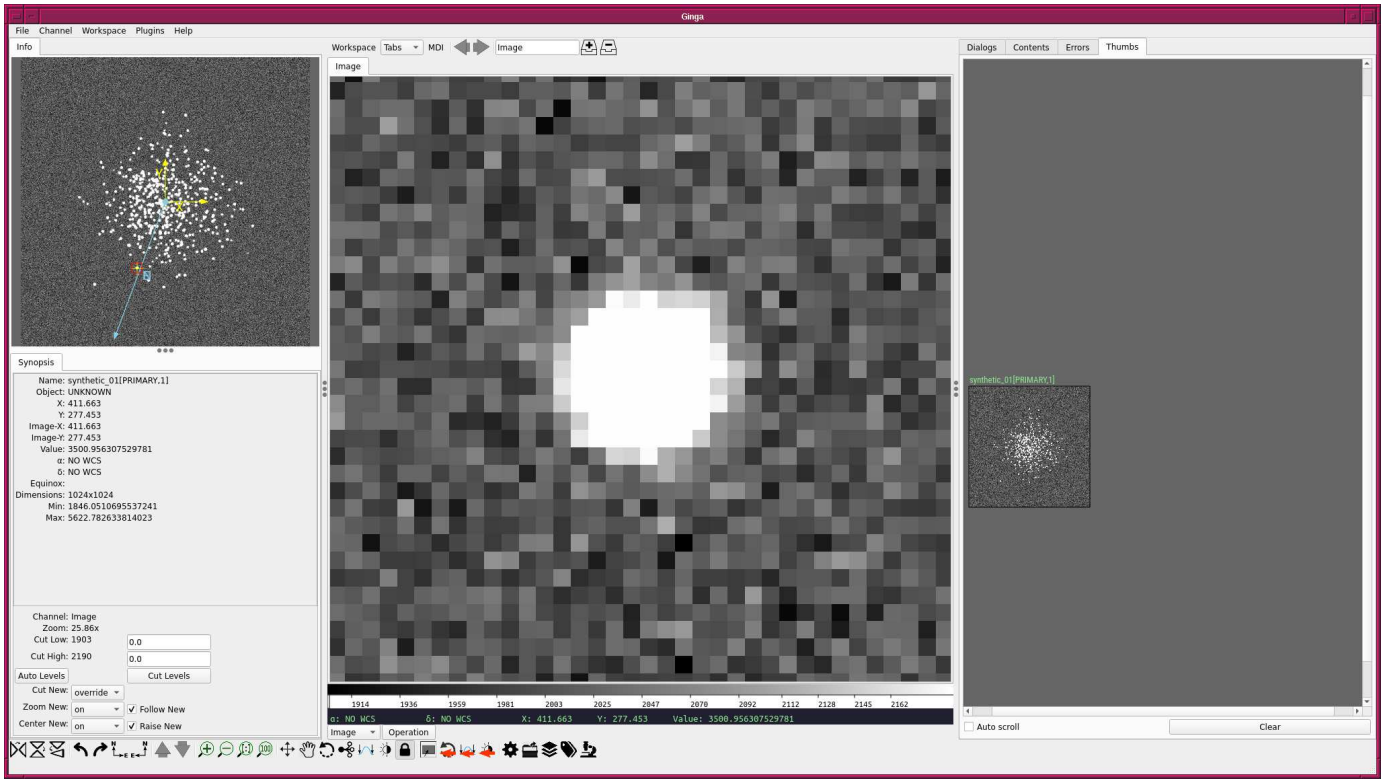

Figure 10: Reading a rough position of selected star.

3.2 Centroid measurement using centre of mass

Make a Python script to carry out centroid measurement using centre of mass.

Python Code 4: advobs202202_s10_04.py

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

#
# Time-stamp: <2022/04/07 15:16:23 (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

# importing photutils module
import photutils.centroids

# constructing parser object
```

```
desc   = 'centroid measurement using centre of mass'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('file', nargs=1, default='', help='input file name')

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

# input parameters
half_width = args.width
x_init     = args.xinit
y_init     = args.yinit
file_fits  = args.file[0]

# making pathlib object
path_fits = pathlib.Path (file_fits)

# checking input FITS file name
if (file_fits == ''):
    # printing message
    print ("You need to specify input file name.")
    # exit
    sys.exit ()
# if input file is not a FITS file, then stop the script
if not (path_fits.suffix == '.fits'):
    # printing message
    print ("Input file must be a FITS file.")
    # exit
    sys.exit ()
# if input file does not exist, then stop the script
if not (path_fits.exists ()):
    # printing message
    print ("Input file does not exist.")
    # exit
    sys.exit ()

# printing information
print ("#")
print ("# input parameters")
print ("#")
print ("#  input file name = %s" % file_fits)
print ("#  half-width of search box = %f" % half_width)
print ("#  x_init = %f" % x_init)
print ("#  y_init = %f" % y_init)
print ("#")

# printing status
print ("# now, reading FITS file...")

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

```
    header = hdu_list[0].header

    # image size
    image_size_x = header['NAXIS1']
    image_size_y = header['NAXIS2']

    # checking x_init and y_init
    if not ( (x_init > 0) and (x_init < image_size_x) ):
        print ("Input x_init value exceed image size.")
        sys.exit ()
    if not ( (y_init > 0) and (y_init < image_size_y) ):
        print ("Input y_init value exceed image size.")
        sys.exit ()

    # reading FITS image data
    data = hdu_list[0].data

# printing status
print ("# finished reading FITS file!")

# printing status
print ("# now, extracting image around the target object...")

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# printing status
print ("# finished extracting image around the target object!")

# printing status
print ("# now, subtracting background...")

# rough background subtraction
subframe -= numpy.median (subframe)

# printing status
print ("# finished subtracting background!")

# printing status
print ("# now, measuring centroid using centre-of-mass...")

# centroid calculation
(x_centre, y_centre) = photutils.centroids.centroid_com (subframe)
x_centre += x_min
y_centre += y_min

# printing status
print ("# finished measuring centroid using centre-of-mass!")

# printing result
print ("#")
print ("# result of the measurement")
print ("#")
```

```
print ("%f %f" % (x_centre, y_centre) )
```

Execute the script, and measure the location of the star.

```
% chmod a+x advobs202202_s10_04.py
% ./advobs202202_s10_04.py -h
usage: advobs202202_s10_04.py [-h] [-w WIDTH] [-x XINIT] [-y YINIT] file

centroid measurement using centre of mass

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target

% ./advobs202202_s10_04.py -w 10 -x 412 -y 277 synthetic_01.fits
#
# input parameters
#
#  input file name = synthetic_01.fits
#  half-width of search box = 10.000000
#  x_init = 412.000000
#  y_init = 277.000000
#
# now, reading FITS file...
# finished reading FITS file!
# now, extracting image around the target object...
# finished extracting image around the target object!
# now, subtracting background...
# finished subtracting background!
# now, measuring centroid using centre-of-mass...
# finished measuring centroid using centre-of-mass!
#
# result of the measurement
#
410.597394 276.800949
```

Measured position is $(x, y) = (410.6, 276.8)$.

3.3 Centroid measurement using 1D Gaussian fitting

Try 1D Gaussian fitting for centroid measurement.

Python Code 5: advobs202202_s10_05.py

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

#
# Time-stamp: <2022/04/07 15:24:12 (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

# importing photutils module
import photutils.centroids

# constructing parser object
desc   = 'centroid measurement using 1-D Gaussian fitting'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('file', nargs=1, default='', help='input file name')

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

# input parameters
half_width = args.width
x_init     = args.xinit
y_init     = args.yinit
file_fits  = args.file[0]

# making pathlib object
path_fits = pathlib.Path (file_fits)

# checking input FITS file name
if (file_fits == ''):
    # printing message
    print ("You need to specify input file name.")
    # exit
    sys.exit ()
# if input file is not a FITS file, then stop the script
if not (path_fits.suffix == '.fits'):
    # printing message
    print ("Input file must be a FITS file.")
    # exit
    sys.exit ()
# if input file does not exist, then stop the script
if not (path_fits.exists ()):
    # printing message
    print ("Input file does not exist.")
```

```
    # exit
    sys.exit ()

# printing information
print ("#")
print ("# input parameters")
print ("#")
print ("#  input file name = %s" % file_fits)
print ("#  half-width of search box = %f" % half_width)
print ("#  x_init = %f" % x_init)
print ("#  y_init = %f" % y_init)
print ("#")

# printing status
print ("# now, reading FITS file...")

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

    # image size
    image_size_x = header['NAXIS1']
    image_size_y = header['NAXIS2']

    # checking x_init and y_init
    if not ( (x_init > 0) and (x_init < image_size_x) ):
        print ("Input x_init value exceed image size.")
        sys.exit ()
    if not ( (y_init > 0) and (y_init < image_size_y) ):
        print ("Input y_init value exceed image size.")
        sys.exit ()

    # reading FITS image data
    data = hdu_list[0].data

# printing status
print ("# finished reading FITS file!")

# printing status
print ("# now, extracting image around the target object...")

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# printing status
print ("# finished extracting image around the target object!")

# printing status
print ("# now, subtracting background...")

# rough background subtraction
subframe -= numpy.median (subframe)
```

```
# printing status
print ("# finished subtracting background!")

# printing status
print ("# now, measuring centroid using centre-of-mass...")

# centroid calculation
#(x_centre, y_centre) = photutils.centroids.centroid_com (subframe)
(x_centre, y_centre) = photutils.centroids.centroid_1dg (subframe)
x_centre += x_min
y_centre += y_min

# printing status
print ("# finished measuring centroid using centre-of-mass!")

# printing result
print ("#")
print ("# result of the measurement")
print ("#")
print ("%f %f" % (x_centre, y_centre) )
```

Run the script.

```
% chmod a+x advobs202202_s10_05.py
% ./advobs202202_s10_05.py -h
usage: advobs202202_s10_05.py [-h] [-w WIDTH] [-x XINIT] [-y YINIT] file

centroid measurement using 1-D Gaussian fitting

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target

% ./advobs202202_s10_05.py -w 10 -x 412 -y 277 synthetic_01.fits
#
# input parameters
#
#  input file name = synthetic_01.fits
#  half-width of search box = 10.000000
#  x_init = 412.000000
#  y_init = 277.000000
#
# now, reading FITS file...
# finished reading FITS file!
# now, extracting image around the target object...
# finished extracting image around the target object!
# now, subtracting background...
# finished subtracting background!
# now, measuring centroid using centre-of-mass...
```

```
# finished measuring centroid using centre-of-mass!
#
# result of the measurement
#
410.781571 276.796606
```

3.4 Centroid measurement using 2D Gaussian fitting

Try 2D Gaussian fitting for centroid measurement.

Python Code 6: advobs202202_s10_06.py

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

#
# Time-stamp: <2022/04/07 15:26:03 (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

# importing photutils module
import photutils.centroids

# constructing parser object
desc   = 'centroid measurement using 2-D Gaussian fitting'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('file', nargs=1, default='', help='input file name')

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

# input parameters
half_width = args.width
x_init     = args.xinit
y_init     = args.yinit
file_fits  = args.file[0]

# making pathlib object
```

```
path_fits = pathlib.Path (file_fits)

# checking input FITS file name
if (file_fits == ''):
    # printing message
    print ("You need to specify input file name.")
    # exit
    sys.exit ()
# if input file is not a FITS file, then stop the script
if not (path_fits.suffix == '.fits'):
    # printing message
    print ("Input file must be a FITS file.")
    # exit
    sys.exit ()
# if input file does not exist, then stop the script
if not (path_fits.exists ()):
    # printing message
    print ("Input file does not exist.")
    # exit
    sys.exit ()

# printing information
print ("#")
print ("# input parameters")
print ("#")
print ("#  input file name = %s" % file_fits)
print ("#  half-width of search box = %f" % half_width)
print ("#  x_init = %f" % x_init)
print ("#  y_init = %f" % y_init)
print ("#")

# printing status
print ("# now, reading FITS file...")

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

    # image size
    image_size_x = header['NAXIS1']
    image_size_y = header['NAXIS2']

    # checking x_init and y_init
    if not ( (x_init > 0) and (x_init < image_size_x) ):
        print ("Input x_init value exceed image size.")
        sys.exit ()
    if not ( (y_init > 0) and (y_init < image_size_y) ):
        print ("Input y_init value exceed image size.")
        sys.exit ()

    # reading FITS image data
    data = hdu_list[0].data

# printing status
print ("# finished reading FITS file!")

# printing status
print ("# now, extracting image around the target object...")
```

```
# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# printing status
print ("# finished extracting image around the target object!")

# printing status
print ("# now, subtracting background...")

# rough background subtraction
subframe -= numpy.median (subframe)

# printing status
print ("# finished subtracting background!")

# printing status
print ("# now, measuring centroid using centre-of-mass...")

# centroid calculation
#(x_centre, y_centre) = photutils.centroids.centroid_com (subframe)
#(x_centre, y_centre) = photutils.centroids.centroid_1dg (subframe)
(x_centre, y_centre) = photutils.centroids.centroid_2dg (subframe)
x_centre += x_min
y_centre += y_min

# printing status
print ("# finished measuring centroid using centre-of-mass!")

# printing result
print ("#")
print ("# result of the measurement")
print ("#")
print ("%f %f" % (x_centre, y_centre) )
```

Run the script.

```
% chmod a+x advobs202202_s10_06.py
% ./advobs202202_s10_06.py -h
usage: advobs202202_s10_06.py [-h] [-w WIDTH] [-x XINIT] [-y YINIT] file

centroid measurement using 2-D Gaussian fitting

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
```

```
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target

% ./advobs202202_s10_06.py -w 10 -x 412 -y 277 synthetic_01.fits
#
# input parameters
#
#  input file name = synthetic_01.fits
#  half-width of search box = 10.000000
#  x_init = 412.000000
#  y_init = 277.000000
#
# now, reading FITS file...
# finished reading FITS file!
# now, extracting image around the target object...
# finished extracting image around the target object!
# now, subtracting background...
# finished subtracting background!
# now, measuring centroid using centre-of-mass...
# finished measuring centroid using centre-of-mass!
#
# result of the measurement
#
410.776759 276.776739
```

3.5 Visualising result of centroid measurement

Make a Python script to carry out centroid measurement and visualising the result.

Python Code 7: advobs202202_s10_07.py

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

#
# Time-stamp: <2022/04/07 15:50:34 (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

# importing photutils module
import photutils.centroids

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

```
# constructing parser object
desc   = 'centroid measurement for a point-source object'
parser = argparse.ArgumentParser (description=desc)

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

# 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 command-line arguments
parser.add_argument ('-c', '--centroid', choices=choices_centroid, \
                     default='com', \
                     help='centroid measurement algorithm (default: com)')
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('-r', '--resolution', type=int, default=450, \
                     help='resolution in DPI (default: 450)')
parser.add_argument ('-m', '--cmap', default='bone', choices=choices_cmap, \
                     help='choice of colour map (default: bone)')
parser.add_argument ('-o', '--output', default='centroid.png', \
                     help='output file name (default: centroid.png)')
parser.add_argument ('file', default='', help='input file name')

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

# input parameters
half_width  = args.width
x_init      = args.xinit
y_init      = args.yinit
centroid    = args.centroid
resolution  = args.resolution
cmap        = args.cmap
file_fits   = args.file
file_output = args.output

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

# checking input FITS file name
if (file_fits == ''):
    # printing message
    print ("You need to specify input file name.")
    # exit
    sys.exit ()
# if input file is not a FITS file, then stop the script
if not (path_fits.suffix == '.fits'):
    # printing message
    print ("Input file must be a FITS file.")
```

```
    # exit
    sys.exit ()
# if input file does not exist, then stop the script
if not (path_fits.exists ()):
    # printing message
    print ("Input file does not exist.")
    # exit
    sys.exit ()
# if output file exists, then stop the script
if (path_output.exists ()):
    # printing message
    print ("Output file exists.")
    # exit
    sys.exit ()
# output file must be either EPS, PDF, PNG, or PS 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 file must be either EPS, PDF, PNG, or PS file.")
    # exit
    sys.exit ()

# printing information
print ("#")
print ("# input parameters")
print ("#")
print ("#  input file name          = %s" % file_fits)
print ("#  half-width of search box = %f" % half_width)
print ("#  x_init                   = %f" % x_init)
print ("#  y_init                   = %f" % y_init)
print ("#  centroid technique       = %s" % centroid)
print ("#  output file name         = %s" % file_output)
print ("#")

# printing status
print ("# now, reading FITS file...")

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

    # image size
    image_size_x = header['NAXIS1']
    image_size_y = header['NAXIS2']

    # checking x_init and y_init
    if not ( (x_init > 0) and (x_init < image_size_x) ):
        print ("Input x_init value exceed image size.")
        sys.exit ()
    if not ( (y_init > 0) and (y_init < image_size_y) ):
        print ("Input y_init value exceed image size.")
        sys.exit ()

    # reading FITS image data
    data = hdu_list[0].data

# printing status
print ("# finished reading FITS file!")
```

```
# printing status
print ("# now, extracting image around the target object...")

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# printing status
print ("# finished extracting image around the target object!")

# printing status
print ("# now, subtracting background...")

# rough background subtraction
subframe -= numpy.median (subframe)

# printing status
print ("# finished subtracting background!")

# printing status
print ("# now, measuring centroid using centre-of-mass...")

# centroid calculation
if (centroid == 'com'):
    (x_centre, y_centre) = photutils.centroids.centroid_com (subframe)
elif (centroid == '1dg'):
    (x_centre, y_centre) = photutils.centroids.centroid_1dg (subframe)
elif (centroid == '2dg'):
    (x_centre, y_centre) = photutils.centroids.centroid_2dg (subframe)
x_centre_sub = x_centre
y_centre_sub = y_centre
x_centre += x_min
y_centre += y_min

# printing status
print ("# finished measuring centroid using centre-of-mass!")

# printing result
print ("#")
print ("# result of the measurement")
print ("#")
print ("%f %f" % (x_centre, y_centre) )

# printing status
print ("# now, generating a plot...")

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

# axes
ax.set_xlabel ('X [pixel]')
```

```
ax.set_ylabel ('Y [pixel]')

# plotting image
im = ax.imshow (subframe, origin='lower', cmap=cmap)
fig.colorbar (im)
ax.plot (x_centre_sub, y_centre_sub, marker='+', color='red', markersize=10)

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

# printing status
print ("# finished generating a plot!")
```

Run the script and generate a PNG file.

```
% chmod a+x advobs202202_s10_07.py
% ./advobs202202_s10_07.py -h
usage: advobs202202_s10_07.py [-h] [-c {com,1dg,2dg}] [-w WIDTH] [-x XINIT]
                              [-y YINIT] [-r RESOLUTION]
                              [-m {viridis,plasma,inferno,magma,cividis,binary,g
ray,bone,pink,spring,summer,autumn,winter,cool,hot,copper,ocean,terrain,gnuplot,
cubehelix,jet,turbo}]
                              [-o OUTPUT]
                              file

centroid measurement for a point-source object

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -c {com,1dg,2dg}, --centroid {com,1dg,2dg}
                        centroid measurement algorithm (default: com)
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target
  -r RESOLUTION, --resolution RESOLUTION
                        resolution 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)
  -o OUTPUT, --output OUTPUT
                        output file name (default: centroid.png)

% ./advobs202202_s10_07.py -c 2dg -w 10 -x 412 -y 277 -m cividis \
? -o synthetic_01_2dg.png synthetic_01.fits
#
# input parameters
#
#  input file name          = synthetic_01.fits
#  half-width of search box = 10.000000
#  x_init                   = 412.000000
```

```
#  y_init                    = 277.000000
#  centroid technique        = 2dg
#  output file name          = synthetic_01_2dg.png
#
# now, reading FITS file...
# finished reading FITS file!
# now, extracting image around the target object...
# finished extracting image around the target object!
# now, subtracting background...
# finished subtracting background!
# now, measuring centroid using centre-of-mass...
# finished measuring centroid using centre-of-mass!
#
# result of the measurement
#
410.776759 276.776739
# now, generating a plot...
# finished generating a plot!
% ls -lF synthetic_01_2dg.png
-rw-r--r--  1 daisuke  taiwan  120677 Apr  7 15:52 synthetic_01_2dg.png
```

Show the PNG file. (Fig. 11)

```
% feh -dF synthetic_01_2dg.png
```

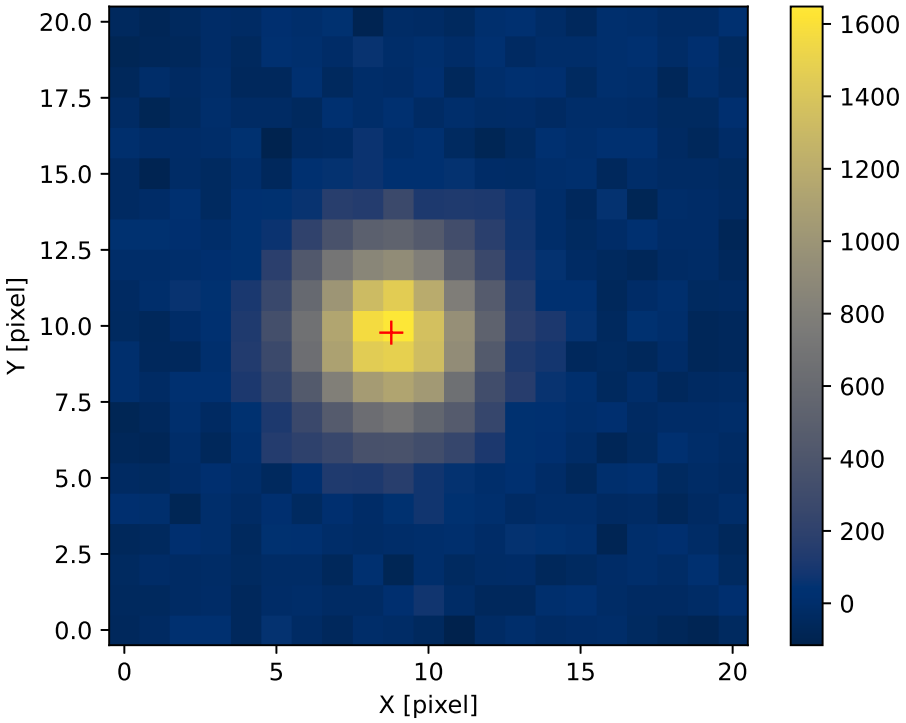

Figure 11: Result of centroid measurement using 2D Gaussian fitting.

4 PSF fitting

Try PSF fitting using `astropy.modeling` module.

4.1 PSF fitting using 2D Gaussian function

Make a Python script to carry out PSF fitting using 2D Gaussian function.

Python Code 8: advobs202202_s10_08.py

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

#
# Time-stamp: <2022/04/07 16:26:36 (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
import astropy.modeling

# importing photutils module
import photutils.centroids

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

# constructing parser object
desc   = 'PSF fitting for a point-source object using 2-D Gaussian profile'
parser = argparse.ArgumentParser (description=desc)

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

# 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 command-line arguments
parser.add_argument ('-c', '--centroid', choices=choices_centroid, \
                     default='com', \
                     help='centroid measurement algorithm (default: com)')
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
```

```
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('-r', '--resolution', type=int, default=450, \
                     help='resolution in DPI (default: 450)')
parser.add_argument ('-m', '--cmap', default='bone', choices=choices_cmap, \
                     help='choice of colour map (default: bone)')
parser.add_argument ('-o', '--output', default='centroid.png', \
                     help='output file name (default: centroid.png)')
parser.add_argument ('file', default='', help='input file name')

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

# input parameters
half_width  = args.width
x_init      = args.xinit
y_init      = args.yinit
centroid    = args.centroid
resolution  = args.resolution
cmap        = args.cmap
file_fits   = args.file
file_output = args.output

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

# checking input FITS file name
if (file_fits == ''):
    # printing message
    print ("You need to specify input file name.")
    # exit
    sys.exit ()
# if input file is not a FITS file, then stop the script
if not (path_fits.suffix == '.fits'):
    # printing message
    print ("Input file must be a FITS file.")
    # exit
    sys.exit ()
# if input file does not exist, then stop the script
if not (path_fits.exists ()):
    # printing message
    print ("Input file does not exist.")
    # exit
    sys.exit ()
# if output file exists, then stop the script
if (path_output.exists ()):
    # printing message
    print ("Output file exists.")
    # exit
    sys.exit ()
# output file must be either EPS, PDF, PNG, or PS 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 file must be either EPS, PDF, PNG, or PS file.")
    # exit
```

```
    sys.exit ()

# printing information
print ("#")
print ("# input parameters")
print ("#")
print ("#  input file name          = %s" % file_fits)
print ("#  half-width of search box = %f" % half_width)
print ("#  x_init                  = %f" % x_init)
print ("#  y_init                  = %f" % y_init)
print ("#  centroid technique      = %s" % centroid)
print ("#  output file name        = %s" % file_output)
print ("#")

# printing status
print ("# now, reading FITS file...")

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

    # image size
    image_size_x = header['NAXIS1']
    image_size_y = header['NAXIS2']

    # checking x_init and y_init
    if not ( (x_init > 0) and (x_init < image_size_x) ):
        print ("Input x_init value exceed image size.")
        sys.exit ()
    if not ( (y_init > 0) and (y_init < image_size_y) ):
        print ("Input y_init value exceed image size.")
        sys.exit ()

    # reading FITS image data
    data = hdu_list[0].data

# printing status
print ("# finished reading FITS file!")

# printing status
print ("# now, extracting image around the target object...")

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# printing status
print ("# finished extracting image around the target object!")

# printing status
print ("# now, subtracting background...")

# rough background subtraction
```

```
subframe -= numpy.median (subframe)

# printing status
print ("# finished subtracting background!")

# printing status
print ("# now, measuring centroid using centre-of-mass...")

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

# PSF fitting
subframe_y, subframe_x = numpy.indices (subframe.shape)
psf_init = astropy.modeling.models.Gaussian2D (x_mean=x_centre, y_mean=y_centre)
fit = astropy.modeling.fitting.LevMarLSQFitter ()
psf_fitted = fit (psf_init, subframe_x, subframe_y, subframe)

x_centre_psf = psf_fitted.x_mean.value
y_centre_psf = psf_fitted.y_mean.value
x_centre_sub = x_centre_psf
y_centre_sub = y_centre_psf
x_centre_psf += x_min
y_centre_psf += y_min
x_fwhm       = psf_fitted.x_fwhm
y_fwhm       = psf_fitted.y_fwhm
fwhm         = (x_fwhm + y_fwhm) / 2.0
amplitude    = psf_fitted.amplitude.value
theta        = psf_fitted.theta.value

# printing status
print ("# finished measuring centroid using centre-of-mass!")

# printing result
print ("#")
print ("# result of the measurement")
print ("#")
print ("#  x_centre  = %f" % x_centre_psf)
print ("#  y_centre  = %f" % y_centre_psf)
print ("#  x_fwhm    = %f" % x_fwhm)
print ("#  y_fwhm    = %f" % y_fwhm)
print ("#  amplitude = %f" % amplitude)
print ("#  theta     = %f" % theta)
print ("#")
print ("# x_fwhm, y_fwhm, fwhm")
print ("%f %f %f" % (x_fwhm, y_fwhm, fwhm) )

# printing status
print ("# now, generating a plot...")

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

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

# plotting image
im = ax.imshow (subframe, origin='lower', cmap=cmap)
fig.colorbar (im)
ax.plot (x_centre_sub, y_centre_sub, marker='+', color='red', markersize=10)

# adding a bar to represent FWHM
bar = matplotlib.patches.Rectangle (xy=(1,1), width=fwhm, height=1.0, \
                                    facecolor='green', edgecolor='white')
ax.add_patch (bar)

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

# printing status
print ("# finished generating a plot!")
```

Execute the script, and measure FWHM (Full-Width at Half-Maximum) of stellar profile.

```
% chmod a+x advobs202202_s10_08.py
% ./advobs202202_s10_08.py -h
usage: advobs202202_s10_08.py [-h] [-c {com,1dg,2dg}] [-w WIDTH] [-x XINIT]
                              [-y YINIT] [-r RESOLUTION]
                              [-m {viridis,plasma,inferno,magma,cividis,binary,g
ray,bone,pink,spring,summer,autumn,winter,cool,hot,copper,ocean,terrain,gnuplot,
cubehelix,jet,turbo}]
                              [-o OUTPUT]
                              file

PSF fitting for a point-source object using 2-D Gaussian profile

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -c {com,1dg,2dg}, --centroid {com,1dg,2dg}
                        centroid measurement algorithm (default: com)
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target
  -r RESOLUTION, --resolution RESOLUTION
                        resolution 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)
  -o OUTPUT, --output OUTPUT
                        output file name (default: centroid.png)

% ./advobs202202_s10_08.py -c 2dg -w 20 -x 412 -y 277 -m cividis \
```

```
? -o synthetic_01_psf.png synthetic_01.fits
#
# input parameters
#
#  input file name          = synthetic_01.fits
#  half-width of search box = 20.000000
#  x_init                   = 412.000000
#  y_init                   = 277.000000
#  centroid technique       = 2dg
#  output file name         = synthetic_01_psf.png
#
# now, reading FITS file...
# finished reading FITS file!
# now, extracting image around the target object...
# finished extracting image around the target object!
# now, subtracting background...
# finished subtracting background!
# now, measuring centroid using centre-of-mass...
# finished measuring centroid using centre-of-mass!
#
# result of the measurement
#
#  x_centre  = 410.776788
#  y_centre  = 276.776960
#  x_fwhm    = 4.961709
#  y_fwhm    = 5.074831
#  amplitude = 1669.599941
#  theta     = -2.817043
#
# x_fwhm, y_fwhm, fwhm
4.961709 5.074831 5.018270
# now, generating a plot...
# finished generating a plot!
% ls -lF *.png
-rw-r--r--  1 daisuke  taiwan  120677 Apr  7 15:52 synthetic_01_2dg.png
-rw-r--r--  1 daisuke  taiwan  120920 Apr  7 16:27 synthetic_01_psf.png
```

Measured FWHM is ~ 5.0 pix and it is consistent with the parameter used for generating the synthetic image. Show the generated plot. The green bar at bottom-left corner indicates FWHM of stellar radial profile. 12

```
% feh -dF synthetic_01_psf.png
```

4.2 PSF fitting using 2D Moffat function

Make a Python script to carry out PSF fitting using 2D Moffat function.

Python Code 9: advobs202202_s10_09.py

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

#
# Time-stamp: <2022/04/07 17:10:32 (CST) daisuke>
#

# importing argparse module
import argparse
```

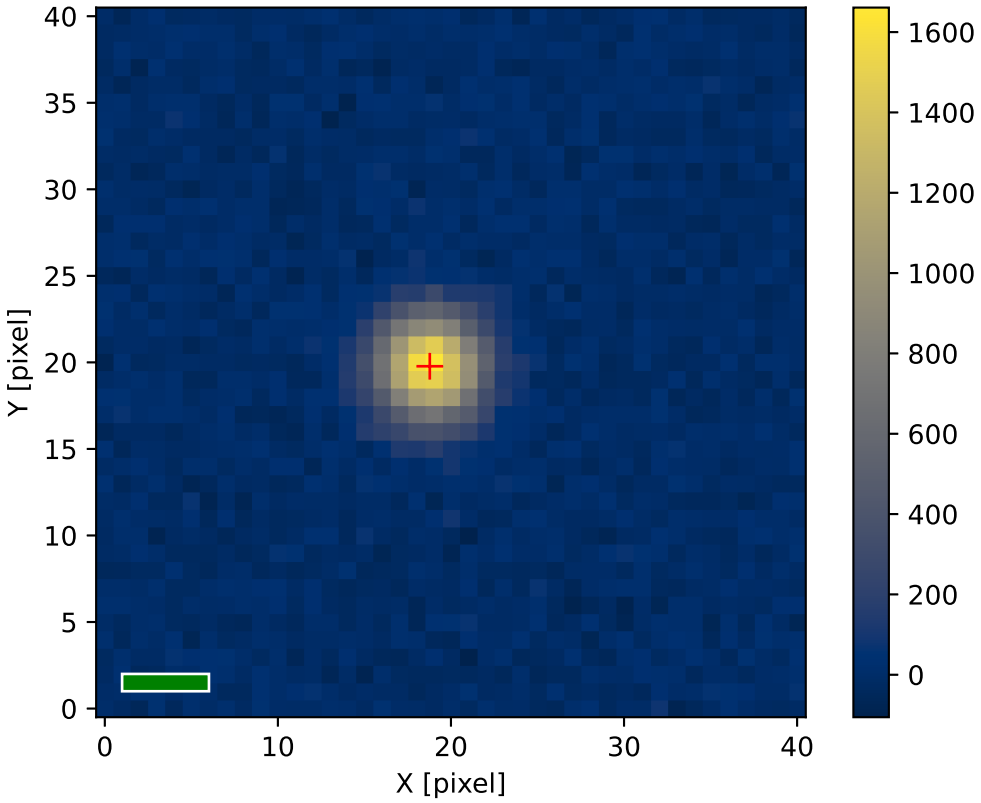

Figure 12: The result of PSF fitting. Measured centre position is shown as a red cross. Derived FWHM of stellar radial profile is shown as a green bar at bottom-left corner of the plot.

```
# importing sys module
import sys

# importing pathlib module
import pathlib

# importing numpy module
import numpy

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

# importing photutils module
import photutils.centroids

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

# constructing parser object
desc   = 'PSF fitting for a point-source object'
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 command-line arguments
parser.add_argument ('-c', '--centroid', choices=choices_centroid, \
                     default='com', \
                     help='centroid measurement algorithm (default: com)')
parser.add_argument ('-p', '--psf', choices=choices_psf, default='2dg', \
                     help='PSF model [2dg=Gaussian, 2dm=Moffat] (default: 2dg)')
parser.add_argument ('-w', '--width', type=int, default=5, \
                     help='half-width of centroid calculation box (default: 5)')
parser.add_argument ('-x', '--xinit', type=int, default=-1, \
                     help='a rough x coordinate of target')
parser.add_argument ('-y', '--yinit', type=int, default=-1, \
                     help='a rough y coordinate of target')
parser.add_argument ('-r', '--resolution', type=int, default=450, \
                     help='resolution in DPI (default: 450)')
parser.add_argument ('-m', '--cmap', default='bone', choices=choices_cmap, \
                     help='choice of colour map (default: bone)')
parser.add_argument ('-o', '--output', default='centroid.png', \
                     help='output file name (default: centroid.png)')
parser.add_argument ('file', default='', help='input file name')

# command-line argument analysis
```

```
args = parser.parse_args ()

# input parameters
half_width  = args.width
x_init      = args.xinit
y_init      = args.yinit
centroid    = args.centroid
psf_model   = args.psf
resolution  = args.resolution
cmap        = args.cmap
file_fits   = args.file
file_output = args.output

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

# checking input FITS file name
if (file_fits == ''):
    # printing message
    print ("You need to specify input file name.")
    # exit
    sys.exit ()
# if input file is not a FITS file, then stop the script
if not (path_fits.suffix == '.fits'):
    # printing message
    print ("Input file must be a FITS file.")
    # exit
    sys.exit ()
# if input file does not exist, then stop the script
if not (path_fits.exists ()):
    # printing message
    print ("Input file does not exist.")
    # exit
    sys.exit ()
# if output file exists, then stop the script
if (path_output.exists ()):
    # printing message
    print ("Output file exists.")
    # exit
    sys.exit ()
# output file must be either EPS, PDF, PNG, or PS 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 file must be either EPS, PDF, PNG, or PS file.")
    # exit
    sys.exit ()

# printing information
print ("#")
print ("# input parameters")
print ("#")
print ("#  input file name          = %s" % file_fits)
print ("#  half-width of search box = %f" % half_width)
print ("#  x_init                   = %f" % x_init)
print ("#  y_init                   = %f" % y_init)
print ("#  centroid technique       = %s" % centroid)
print ("#  output file name         = %s" % file_output)
```

```
print ("#")

# printing status
print ("# now, reading FITS file...")

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

    # image size
    image_size_x = header['NAXIS1']
    image_size_y = header['NAXIS2']

    # checking x_init and y_init
    if not ( (x_init > 0) and (x_init < image_size_x) ):
        print ("Input x_init value exceed image size.")
        sys.exit ()
    if not ( (y_init > 0) and (y_init < image_size_y) ):
        print ("Input y_init value exceed image size.")
        sys.exit ()

    # reading FITS image data
    data = hdu_list[0].data

# printing status
print ("# finished reading FITS file!")

# printing status
print ("# now, extracting image around the target object...")

# region of calculation
x_min = x_init - half_width
x_max = x_init + half_width + 1
y_min = y_init - half_width
y_max = y_init + half_width + 1

# extraction of subframe for calculation
subframe = data[y_min:y_max, x_min:x_max]

# printing status
print ("# finished extracting image around the target object!")

# printing status
print ("# now, subtracting background...")

# rough background subtraction
subframe -= numpy.median (subframe)

# printing status
print ("# finished subtracting background!")

# printing status
print ("# now, measuring centroid...")

# centroid calculation
if (centroid == 'com'):
    (x_centre, y_centre) = photutils.centroids.centroid_com (subframe)
elif (centroid == '1dg'):
```

```
    (x_centre, y_centre) = photutils.centroids.centroid_1dg (subframe)
elif (centroid == '2dg'):
    (x_centre, y_centre) = photutils.centroids.centroid_2dg (subframe)

# printing status
print ("# finished measuring!")

# printing status
print ("# now, measuring PSF...")

# PSF fitting
subframe_y, subframe_x = numpy.indices (subframe.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)

# result of fitting
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 + x_min
    y_centre_psf = psf_fitted.y_mean.value + y_min
    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 + x_min
    y_centre_psf = psf_fitted.y_0.value + y_min
    alpha        = psf_fitted.alpha.value
    gamma        = psf_fitted.gamma.value
    fwhm         = psf_fitted.fwhm

# printing status
print ("# finished measuring PSF!")

# printing result
print ("#")
print ("# result of the measurement")
print ("#")
print ("#  x_centre  = %f" % x_centre_psf)
print ("#  y_centre  = %f" % y_centre_psf)
print ("#  amplitude = %f" % amplitude)
if (psf_model == '2dg'):
    print ("#  x_fwhm    = %f" % x_fwhm)
    print ("#  y_fwhm    = %f" % y_fwhm)
    print ("#  theta     = %f" % theta)
elif (psf_model == '2dm'):
    print ("#  alpha     = %f" % alpha)
    print ("#  gamma     = %f" % gamma)
```

```
    print ("#  fwhm       = %f" % fwhm)
print ("#")
print ("# X_CENTRE, Y_CENTRE, FWHM")
print ("%f %f %f" % (x_centre_psf, y_centre_psf, fwhm) )

# printing status
print ("# now, generating a plot...")

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

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

# plotting image
im = ax.imshow (subframe, origin='lower', cmap=cmap)
fig.colorbar (im)
ax.plot (x_centre_sub, y_centre_sub, marker='+', color='red', markersize=10)

# adding a bar to represent FWHM
bar = matplotlib.patches.Rectangle (xy=(1,1), width=fwhm, height=1.0, \
                                    facecolor='green', edgecolor='white')
ax.add_patch (bar)

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

# printing status
print ("# finished generating a plot!")
```

Run the script.

```
% chmod a+x ./advobs202202_s10_09.py
% ./advobs202202_s10_09.py -h
usage: advobs202202_s10_09.py [-h] [-c {com,1dg,2dg}] [-p {2dg,2dm}]
                              [-w WIDTH] [-x XINIT] [-y YINIT] [-r RESOLUTION]
                              [-m {viridis,plasma,inferno,magma,cividis,binary,g
ray,bone,pink,spring,summer,autumn,winter,cool,hot,copper,ocean,terrain,gnuplot,
cubehelix,jet,turbo}]
                              [-o OUTPUT]
                              file

PSF fitting for a point-source object

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -c {com,1dg,2dg}, --centroid {com,1dg,2dg}
                        centroid measurement algorithm (default: com)
  -p {2dg,2dm}, --psf {2dg,2dm}
                        PSF model [2dg=Gaussian, 2dm=Moffat] (default: 2dg)
  -w WIDTH, --width WIDTH
                        half-width of centroid calculation box (default: 5)
```

```
  -x XINIT, --xinit XINIT
                        a rough x coordinate of target
  -y YINIT, --yinit YINIT
                        a rough y coordinate of target
  -r RESOLUTION, --resolution RESOLUTION
                        resolution 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)
  -o OUTPUT, --output OUTPUT
                        output file name (default: centroid.png)

% ./advobs202202_s10_09.py -c com -p 2dm -w 20 -x 412 -y 277 -m cividis \
? -o synthetic_01_psf2.png synthetic_01.fits
#
# input parameters
#
#  input file name          = synthetic_01.fits
#  half-width of search box = 20.000000
#  x_init                   = 412.000000
#  y_init                   = 277.000000
#  centroid technique       = com
#  output file name         = synthetic_01_psf2.png
#
# now, reading FITS file...
# finished reading FITS file!
# now, extracting image around the target object...
# finished extracting image around the target object!
# now, subtracting background...
# finished subtracting background!
# now, measuring centroid...
# finished measuring!
# now, measuring PSF...
# finished measuring PSF!
#
# result of the measurement
#
#  x_centre  = 410.776551
#  y_centre  = 276.777662
#  amplitude = 1675.196467
#  alpha     = 74.484154
#  gamma     = 25.833003
#  fwhm      = 4.995706
#
# X_CENTRE, Y_CENTRE, FWHM
410.776551 276.777662 4.995706
# now, generating a plot...
# finished generating a plot!
% ls -lF *.png
-rw-r--r--  1 daisuke  taiwan  120677 Apr  7 15:52 synthetic_01_2dg.png
-rw-r--r--  1 daisuke  taiwan  120920 Apr  7 16:27 synthetic_01_psf.png
-rw-r--r--  1 daisuke  taiwan  122017 Apr  7 17:12 synthetic_01_psf2.png
```

Again, we have obtained FWHM of ~ 5 pix as expected.
Show the PNG image. (Fig. 13)

```
% feh -dF synthetic_01_moffat.png
```

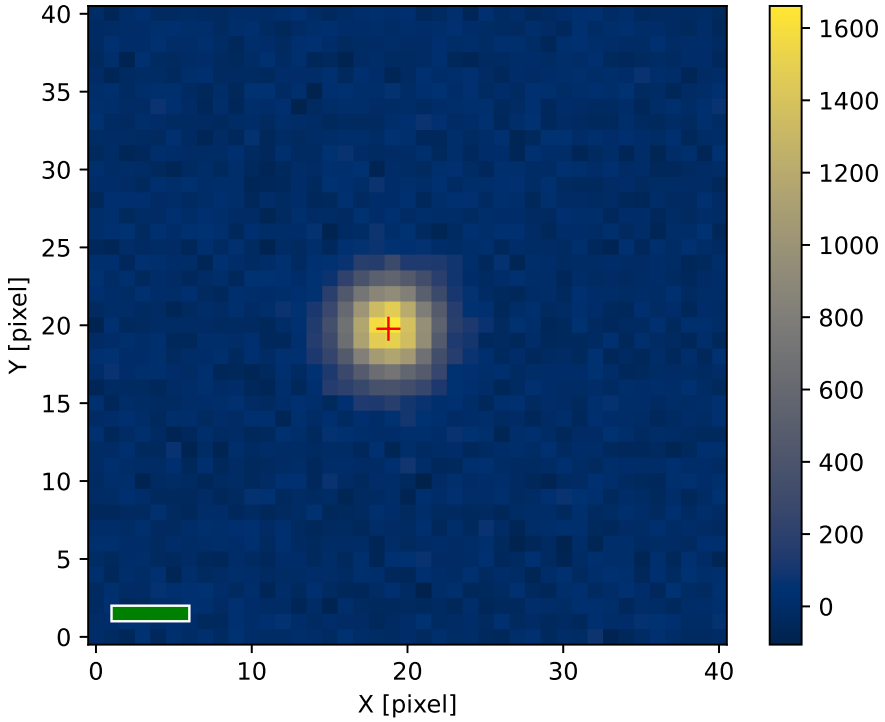

Figure 13: Result of PSF fitting using 2D Moffat function.

5 For your further reading

1. Read chapter 5 of "Handbook of CCD Astronomy" and learn about image centring and two-dimensional profile fitting.
   - Handbook of CCD Astronomy (2nd Edition)
     - Steve B. Howell
     - Cambridge University Press
     - `https://doi.org/10.1017/CBO9780511807909`
2. Read a paper "Basic Photometry Techniques" and learn about image centring.
   - "Basic Photometry Techniques"
     - G. S. Da Costa
     - "Astronomical CCD Observing and Reduction Techniques", page 90
     - 1992
     - `http://www.aspbooks.org/a/volumes/article_details/?paper_id=7035`

6 Exercise

1. Describe a method to measure the centre of the image of a star. Show mathematical formulae.
2. What is PSF (Point-Spread Function)? Which functions are often used to describe stellar PSF? Show mathematical formulae of those functions.

3. What is FWHM (Full-Width at Half-Maximum)? How FWHM is used for astronomy?

4. Consider a Gaussian distribution of the form

$$f_G(x) = \frac{a}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] + b.$$

   Show that FWHM can be written as FWHM $= 2\sqrt{2\log 2}\sigma$.

5. What is Moffat function? Show mathematical formula of 2-dimensional Moffat function. Plot a Moffat function. Plot and compare 1-dimensional Gaussian function and 1-dimensional Moffat function.

6. Make your own Python script to produce a synthetic image of a galaxy cluster. Describe the design of your Python script. Show the source code of your Python script. Run the script, and generate a FITS file. Use Ginga to visualise the FITS file and show the image.

7. Select a FITS file of reduced (dark-subtracted and flatfielded) object frame from the session 08 "Basic CCD Data Reduction".

   (a) Which file have you selected?

   (b) Use Ginga to show the image. Choose a star for your measurement. Mark a star with a red circle using Ginga. Export an image. Show the image.

   (c) Make your own Python script to measure centroid of the star. Describe the design of your program. Show the source code of your Python script. Execute the script, and show the result.

   (d) Make your own Python script to carry out PSF fitting of the star. Describe the design of your program. Show the source code of your Python script. Execute the script, and show the result.

8. Make your own Python script to construct a radial profile of a star on the image and carry out fitting using Gaussian function or Moffat function. Describe the design of your program. Show the source code of your Python script. Execute the script, and show the result.