Advanced Astronomical Observations 2022
Session 11: Aperture Photometry

Kinoshita Daisuke

15 April 2022
publicly accessible version

Aperture photometry is a simple and effective way to measure the brightness of point-like sources on astronomical images. For this session, we try aperture photometry using `photutils` package.

1 Photutils package

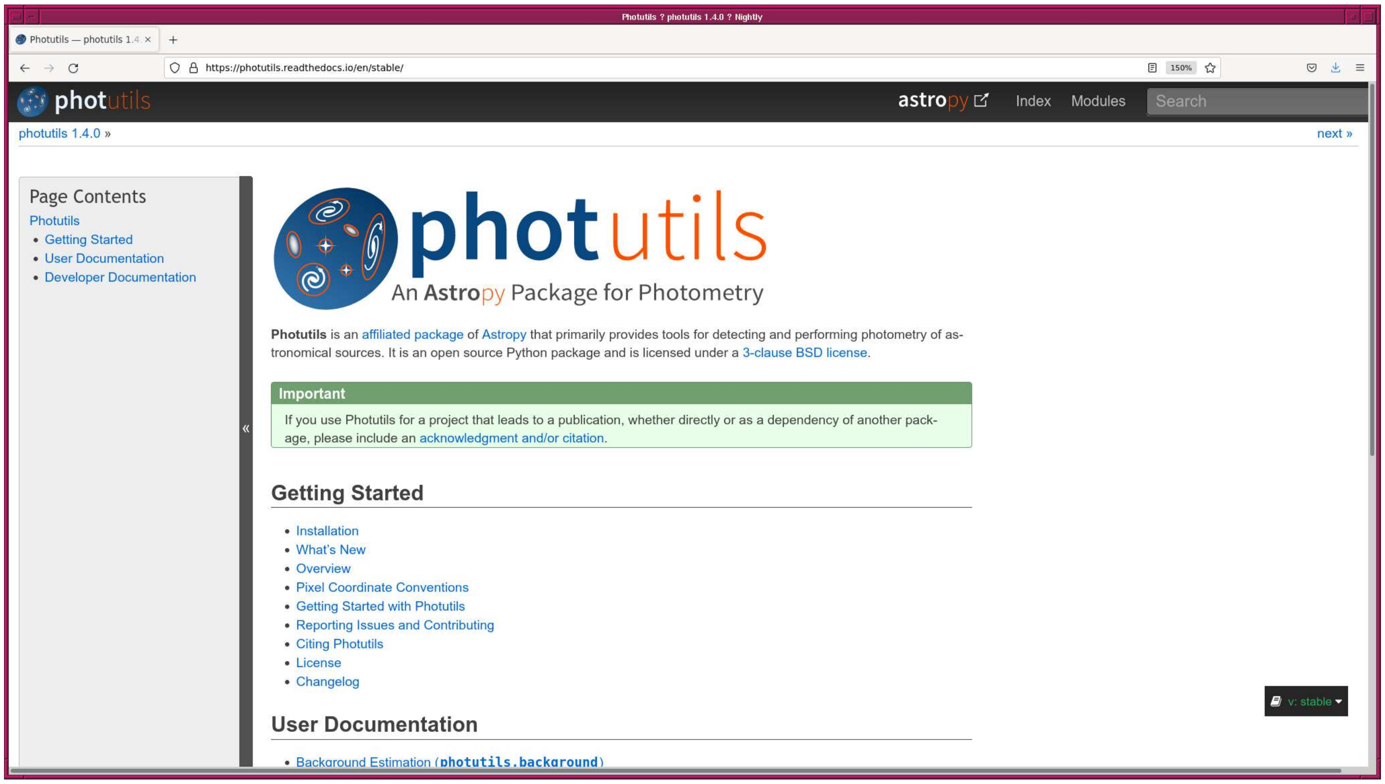
For this session, `photutils` package is needed. Visit following websites to learn about `photutils`.

- photutils
  - `https://photutils.readthedocs.io/` (Fig. 1)
  - `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.



Photutils is an affiliated package of Astropy that primarily provides tools for detecting and performing photometry of astronomical sources. It is an open source Python package and is licensed under a 3-clause BSD license.

Important

If you use Photutils for a project that leads to a publication, whether directly or as a dependency of another package, please include an acknowledgment and/or citation.

Getting Started

- Installation
- What's New
- Overview
- Pixel Coordinate Conventions
- Getting Started with Photutils
- Reporting Issues and Contributing
- Citing Photutils
- License
- Changelog

User Documentation

- Background Estimation (`photutils.background`)

Figure 1: 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 (Fig. 2), read the installation instruction, and install the package on your computer. Pay attention to the requirements for installation of `photutils` package. You need to install all the required software on your computer before starting the installation of `photutils`. Check the list of required software and make sure to examine whether you have required software on your computer one by one.

Read the user documentation of `photutils` to learn about usage of the package. (3)

2 Generating synthetic images

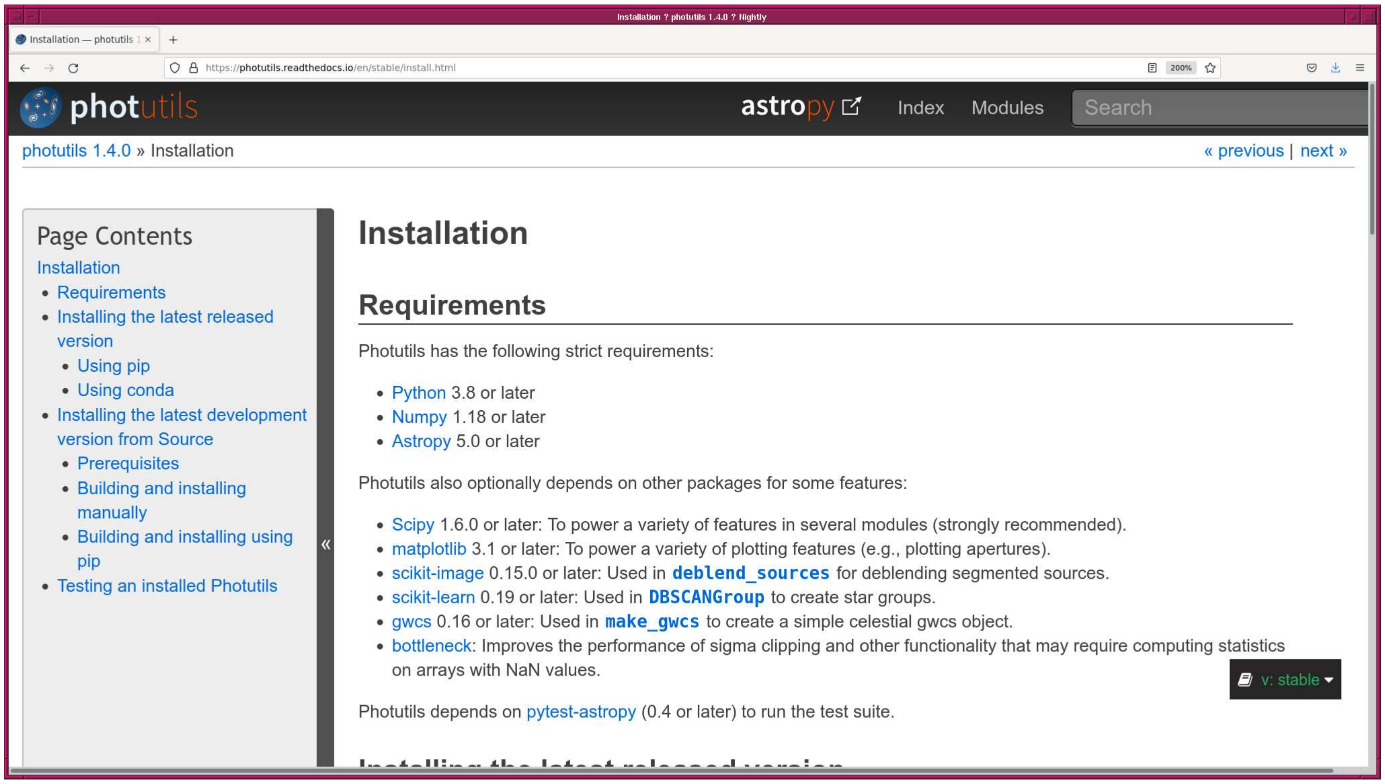
Use `photutils.datasets` to generate a synthetic image for aperture photometry.

2.1 Making a synthetic image for aperture photometry

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

Python Code 1: advobs202202_s11_01.py

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



Installation ? photutils 1.4.0 ? Nightly
Installation — photutils 1
https://photutils.readthedocs.io/en/stable/install.html
200%
photutils
astropy
Index
Modules
Search
v: stable

photutils 1.4.0 » Installation « previous | next »

Installation

Requirements

Photutils has the following strict requirements:

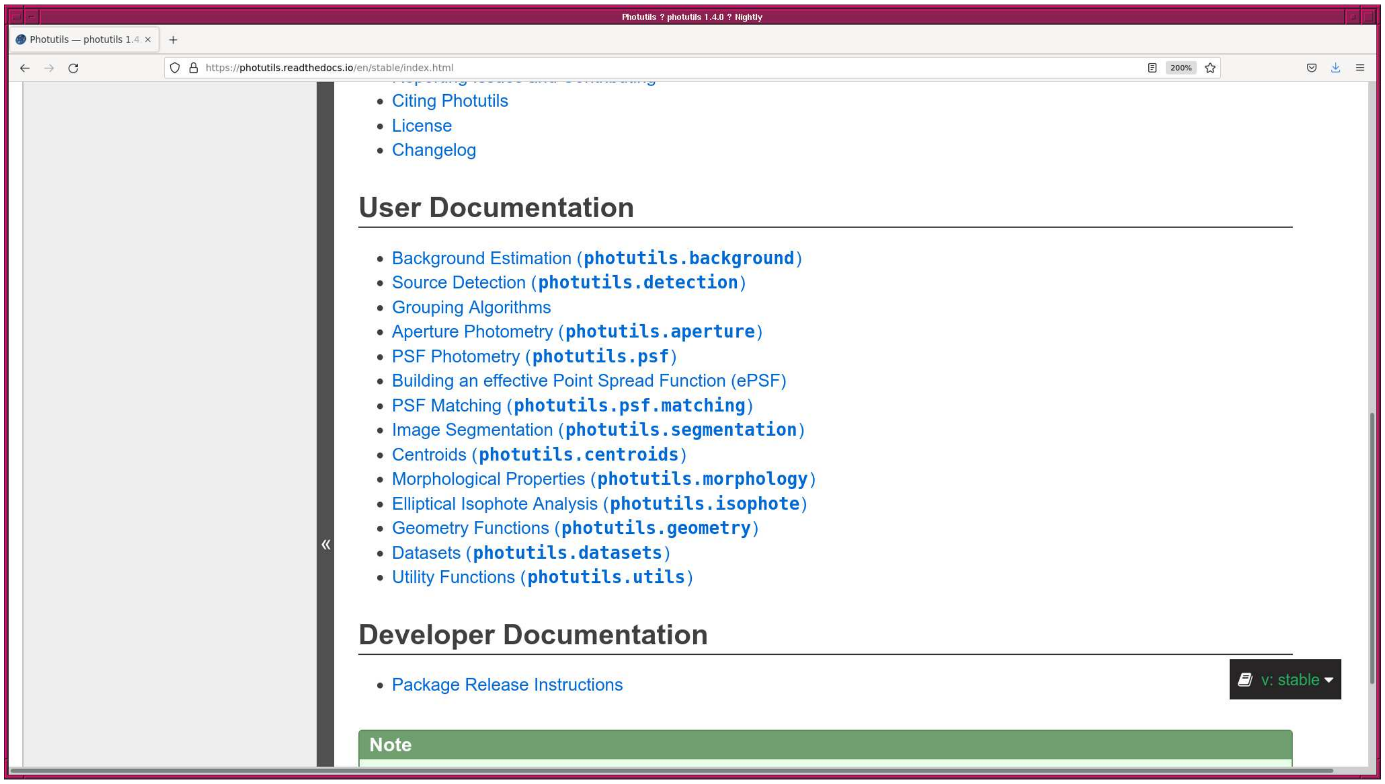
- Python 3.8 or later
- Numpy 1.18 or later
- Astropy 5.0 or later

Photutils also optionally depends on other packages for some features:

- Scipy 1.6.0 or later: To power a variety of features in several modules (strongly recommended).
- matplotlib 3.1 or later: To power a variety of plotting features (e.g., plotting apertures).
- scikit-image 0.15.0 or later: Used in `deblend_sources` for deblending segmented sources.
- scikit-learn 0.19 or later: Used in `DBSCANGroup` to create star groups.
- gwcs 0.16 or later: Used in `make_gwcs` to create a simple celestial gwcs object.
- bottleneck: Improves the performance of sigma clipping and other functionality that may require computing statistics on arrays with NaN values.

Photutils depends on pytest-astropy (0.4 or later) to run the test suite.

Figure 2: The installation instruction and description of the requirements at the official website of `photutils` package at `https://photutils.readthedocs.io/en/stable/install.html`.



Photutils ? photutils 1.4.0 ? Nightly
Photutils — photutils 1.4
https://photutils.readthedocs.io/en/stable/index.html
200%
v: stable
Note

- Citing Photutils
- License
- Changelog

User Documentation

- Background Estimation (`photutils.background`)
- Source Detection (`photutils.detection`)
- Grouping Algorithms
- Aperture Photometry (`photutils.aperture`)
- PSF Photometry (`photutils.psf`)
- Building an effective Point Spread Function (ePSF)
- PSF Matching (`photutils.psf.matching`)
- Image Segmentation (`photutils.segmentation`)
- Centroids (`photutils.centroids`)
- Morphological Properties (`photutils.morphology`)
- Elliptical Isophote Analysis (`photutils.isophote`)
- Geometry Functions (`photutils.geometry`)
- Datasets (`photutils.datasets`)
- Utility Functions (`photutils.utils`)

Developer Documentation

- Package Release Instructions

Figure 3: The user documentation of `photutils` on the official website of `photutils` package at `https://photutils.readthedocs.io/`.

```
#
# Time-stamp: <2022/04/14 11:56:57 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing datetime module
import datetime

# importing numpy module
import numpy

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

# importing photutils module
import photutils.datasets

# constructing parser object
desc   = 'generating a synthetic image of sky background and stars'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-b', '--background', type=float, default=2500.0, \
                     help='background level (default: 2500)')
parser.add_argument ('-s', '--sigma', type=float, default=50.0, \
                     help='noise level (default: 50)')
parser.add_argument ('-n', '--nstar', type=int, default=5, \
                     help='number of stars to generate (n x n) (default: 5)')
parser.add_argument ('-f1', '--fluxmin', type=float, default=100000.0, \
                     help='min total flux of star (default: 100000)')
parser.add_argument ('-f2', '--fluxmax', type=float, default=200000.0, \
                     help='max total flux of star (default: 200000)')
parser.add_argument ('-p1', '--psfmin', type=float, default=4.0, \
                     help='min FWHM of stellar radial profile (default: 4)')
parser.add_argument ('-p2', '--psfmax', type=float, default=6.0, \
                     help='max FWHM of stellar radial profile (default: 6)')
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_total_min       = args.fluxmin
flux_total_max       = args.fluxmax
psf_fwhm_min         = args.psfmin
psf_fwhm_max         = args.psfmax
```

```
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 ()
if not (path_output.suffix == '.fits'):
    # printing message
    print ("Output file must be a FITS file.")
    # exit
    sys.exit ()
# existence check
if (path_output.exists ()):
    # printing message
    print ("Output file exists.")
    # exit
    sys.exit ()

# image size
image_size_x = 1024
image_size_y = 1024
image_size   = (image_size_x, image_size_y)
coord_min    = 100.0
coord_max    = 900.0

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

# command name
command = sys.argv[0]

# printing status
print ("Now, generating synthetic sky background...")

# 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 synthetic sky background!")

# printing status
print ("Now, generating synthetic stars...")

# grid of data for generating synthetic stars
#
# x coord ==> [100, 300, 500, 700, 900, 100, 300, 500, 700, 900, 100, ...]
# y coord ==> [100, 100, 100, 100, 100, 300, 300, 300, 300, 300, 500, ...]
#

# initialisation of numpy arrays
```

```
list_x    = numpy.array ([])
list_y    = numpy.array ([])
list_flux = numpy.array ([])
list_psf  = numpy.array ([])

# generating X coordinates of synthetic stars
for i in range (nstar):
    list_x \
        = numpy.append (list_x, numpy.linspace (coord_min, coord_max, nstar) )

# generating X coordinates of synthetic stars
for y in numpy.linspace (coord_min, coord_max, nstar):
    list_y = numpy.append (list_y, numpy.repeat (y, nstar) )

# flux of stars
for i in range (nstar):
    list_flux \
        = numpy.append (list_flux, \
                        numpy.linspace (flux_total_min, flux_total_max, nstar) )

# FWHM of PSF of synthetic stars
for psf in numpy.linspace (psf_fwhm_min, psf_fwhm_max, nstar):
    list_psf = numpy.append (list_psf, numpy.repeat (psf, nstar) )

# 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']   = list_x
source_table['y_0']   = list_y
source_table['flux']  = list_flux
source_table['sigma'] = list_psf / fwhm_sigma

# printing source table
print ("source_table:")
print (source_table)

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

# printing status
print ("Finished generating synthetic stars!")

# printing status
print ("Now, making a synthetic image skybg + stars...")

# making synthetic image by adding background and stars
image = image_background + image_star

# printing status
print ("Finished making a synthetic image skybg + stars!")

# printing status
print ("Now, preparing 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 skybg + stars"
header['comment'] = "Options given:"
header['comment'] = "  image size           = %d x %d" % (image_size)
header['comment'] = "  sky background level = %f ADU" % (sky_background_level)
header['comment'] = "  noise level          = %f ADU" % (noise_level)
header['comment'] = "  number of stars      = %d x %d" % (nstar, nstar)
header['comment'] = "  min flux of stars    = %f ADU" % (flux_total_min)
header['comment'] = "  max flux of stars    = %f ADU" % (flux_total_max)
header['comment'] = "  min FWHM of stars    = %f ADU" % (psf_fwhm_min)
header['comment'] = "  max FWHM of stars    = %f ADU" % (psf_fwhm_max)

# printing status
print ("Finished preparing FITS header!")

# printing status
print ("Now, writing data into FITS file \"%s\"..." % file_output)

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

# printing status
print ("Finished writing data into FITS file \"%s\"!" % file_output)
```

Execute the script and generate a synthetic image.

```
% chmod a+x advobs202202_s11_01.py
% ./advobs202202_s11_01.py -h
usage: advobs202202_s11_01.py [-h] [-b BACKGROUND] [-s SIGMA] [-n NSTAR]
                              [-f1 FLUXMIN] [-f2 FLUXMAX] [-p1 PSFMIN]
                              [-p2 PSFMAX] [-o OUTPUT]

generating a synthetic image of sky background and stars

optional arguments:
  -h, --help            show this help message and exit
  -b BACKGROUND, --background BACKGROUND
                        background level (default: 2500)
  -s SIGMA, --sigma SIGMA
                        noise level (default: 50)
  -n NSTAR, --nstar NSTAR
                        number of stars to generate (n x n) (default: 5)
  -f1 FLUXMIN, --fluxmin FLUXMIN
                        min total flux of star (default: 100000)
  -f2 FLUXMAX, --fluxmax FLUXMAX
                        max total flux of star (default: 200000)
  -p1 PSFMIN, --psfmin PSFMIN
                        min FWHM of stellar radial profile (default: 4)
  -p2 PSFMAX, --psfmax PSFMAX
                        max FWHM of stellar radial profile (default: 6)
  -o OUTPUT, --output OUTPUT
                        output file name

% ./advobs202202_s11_01.py -b 6400 -s 80 -p1 3 -p2 7 -f1 100000 -f2 500000 \
? -o synstars_0.fits
Now, generating synthetic sky background...
```

```
Finished generating synthetic sky background!
Now, generating synthetic stars...
source_table:
 x_0   y_0    flux          sigma
----- ----- -------- ------------------
100.0 100.0 100000.0 1.2739827004320285
300.0 100.0 200000.0 1.2739827004320285
500.0 100.0 300000.0 1.2739827004320285
700.0 100.0 400000.0 1.2739827004320285
900.0 100.0 500000.0 1.2739827004320285
100.0 300.0 100000.0 1.6986436005760381
300.0 300.0 200000.0 1.6986436005760381
500.0 300.0 300000.0 1.6986436005760381
700.0 300.0 400000.0 1.6986436005760381
900.0 300.0 500000.0 1.6986436005760381
100.0 500.0 100000.0 2.1233045007200477
300.0 500.0 200000.0 2.1233045007200477
500.0 500.0 300000.0 2.1233045007200477
700.0 500.0 400000.0 2.1233045007200477
900.0 500.0 500000.0 2.1233045007200477
100.0 700.0 100000.0  2.547965400864057
300.0 700.0 200000.0  2.547965400864057
500.0 700.0 300000.0  2.547965400864057
700.0 700.0 400000.0  2.547965400864057
900.0 700.0 500000.0  2.547965400864057
100.0 900.0 100000.0  2.972626301008067
300.0 900.0 200000.0  2.972626301008067
500.0 900.0 300000.0  2.972626301008067
700.0 900.0 400000.0  2.972626301008067
900.0 900.0 500000.0  2.972626301008067
Finished generating synthetic stars!
Now, making a synthetic image skybg + stars...
Finished making a synthetic image skybg + stars!
Now, preparing FITS header...
Finished preparing FITS header!
Now, writing data into FITS file "synstars_0.fits"...
Finished writing data into FITS file "synstars_0.fits"!
% ls -lF
total 9
-rwxr-xr-x  1 daisuke  taiwan     6409 Apr 14 11:56 advobs202202_s11_01.py*
-rw-r--r--  1 daisuke  taiwan     4148 Apr 14 11:30 advobs202202_s11_01.py~
-rw-r--r--  1 daisuke  taiwan  8392320 Apr 14 12:03 synstars_0.fits
% file synstars_0.fits
synstars_0.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_s11_02.py

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

#
# Time-stamp: <2022/04/14 12:09:23 (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
list_rejection = ['NONE', 'sigclip']
parser.add_argument ('-r', '--rejection', default='NONE', \
                     choices=list_rejection, \
                     help='rejection algorithm (default: NONE)')
parser.add_argument ('-t', '--threshold', type=float, default=4.0, \
                     help='threshold for sigma clipping (default: 4.0)')
parser.add_argument ('-n', '--maxiters', type=int, default=10, \
                     help='maximum number of iterations (default: 10)')
parser.add_argument ('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 ("### ERROR: file '%s' is not a FITS file!" % file_fits)
        # skipping
```

```
            continue
        # if the file does not exist, then skip
        if not (path_fits.exists ()):
            # printing message
            print ("### ERROR: file '%s' does not exist!" % file_fits)
            # skipping
            continue

        # opening FITS file
        with astropy.io.fits.open (file_fits) as hdu_list:
            # header of primary HDU
            header0 = hdu_list[0].header
            # 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_s11_02.py
% ./advobs202202_s11_02.py -h
usage: advobs202202_s11_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_s11_02.py synstars_0.fits
# Input parameters
#   rejection algorithm = NONE
#
# file name                      npix     mean   median  stddev      min      max
#
synstars_0.fits               1048576  6407.04  6400.31  254.80  6030.68 52932.45
% ./advobs202202_s11_02.py -r sigclip synstars_0.fits
# Input parameters
#   rejection algorithm = sigclip
#   threshold of sigma-clipping = 4.000000
#   maximum number of iterations = 10
#
# file name                      npix     mean   median  stddev      min      max
#
synstars_0.fits               1046063  6400.11  6400.08   80.14  6080.15  6720.57
```

The mean pixel value is ~ 6400 ADU as expected.

2.3 Visual inspection of the image

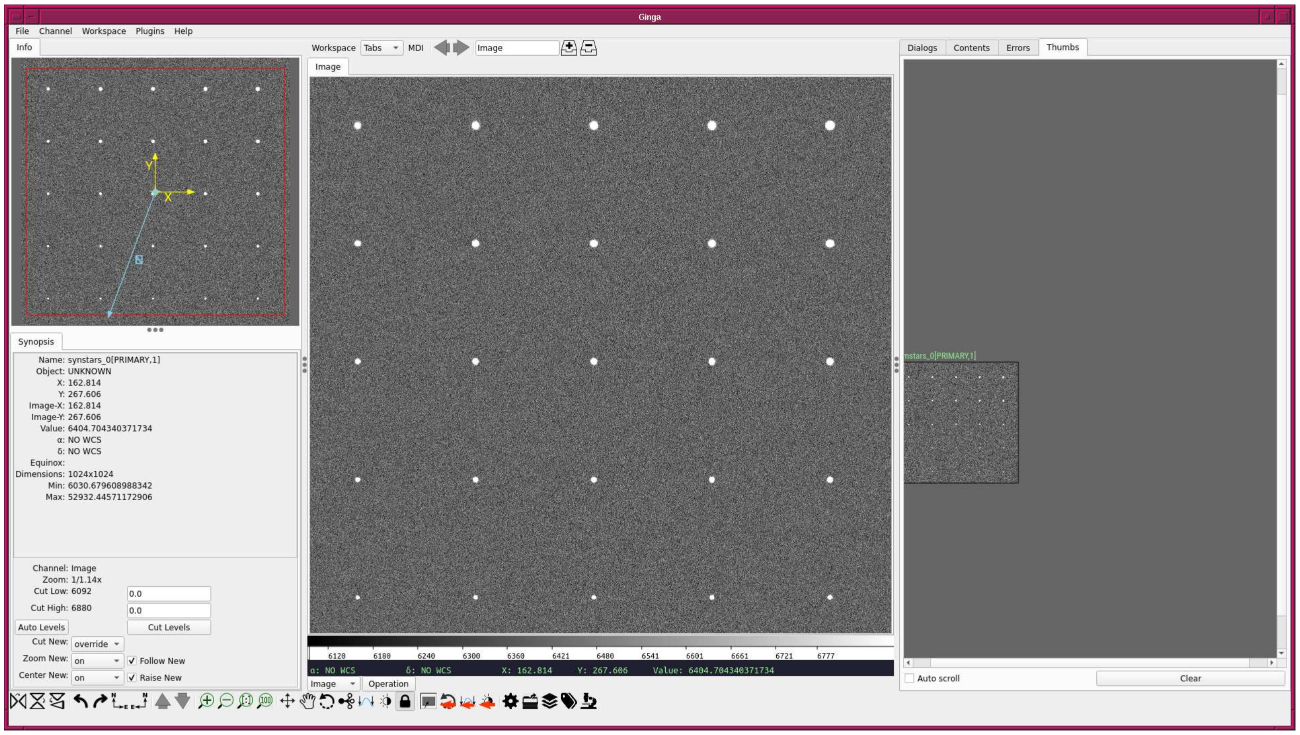
Use Ginga to show the synthetic image simulating sky background and stars. 25 stars are seen on the image. (Fig. 4)

```
% ginga synstars_0.fits
```

3 Centroid measurements

3.1 Knowing a rough position of a star using Ginga

Use Ginga to know a rough position of a star.



Name: synstars_0[PRIMARY,1]
Object: UNKNOWN
X: 162.814
Y: 267.606
Image-X: 162.814
Image-Y: 267.606
Value: 6404.704340371734
α: NO WCS
δ: NO WCS
Equinox:
Dimensions: 1024x1024
Min: 6030.679608988342
Max: 52932.44571172906

Channel: Image
Zoom: 1/1.14x
Cut Low: 6092 0.0
Cut High: 6880 0.0
Auto Levels Cut Levels
Cut New: override
Zoom New: on ✓ Follow New
Center New: on ✓ Raise New

```
α: NO WCS     δ: NO WCS     X: 162.814     Y: 267.606     Value: 6404.704340371734
```

Figure 4: The synthetic image simulating sky background and stars generated by photutils.datasets module.

1. Start Ginga.
2. Load the image "`synstars_0.fits`".
3. Choose a star for your centroid measurement.
4. Zoom in and pan to the star you have picked. (Fig. 5)
5. Visit the menu "Plugins".
6. Choose "Analysis, and then click the menu "Pick". (Fig. 6)
7. Move your mouse cursor to your target.
8. Click the left button of your mouse. (Fig. 7)
9. Read values of (x, y) coordinate of your target fro the panel on the right hand side of the window.

For example, a star at the centre of the image gives the coordinate (x, y) ∼ (501, 501).

Actually, the coordinate shown by IRAF, SAOimage DS9, Ginga, etc. are different from the coordinate used by Python/Numpy. The coordinate used by IRAF, SAOimage DS9, Ginga, etc. starts from 1, but the coordinate used by Python/Numpy starts from 0. Therefore, The coordinate we actually need is (x, y) ∼ (500, 500). But, don't worry, it's usually no problem using the value shown by Ginga directly. For more information, read following document.

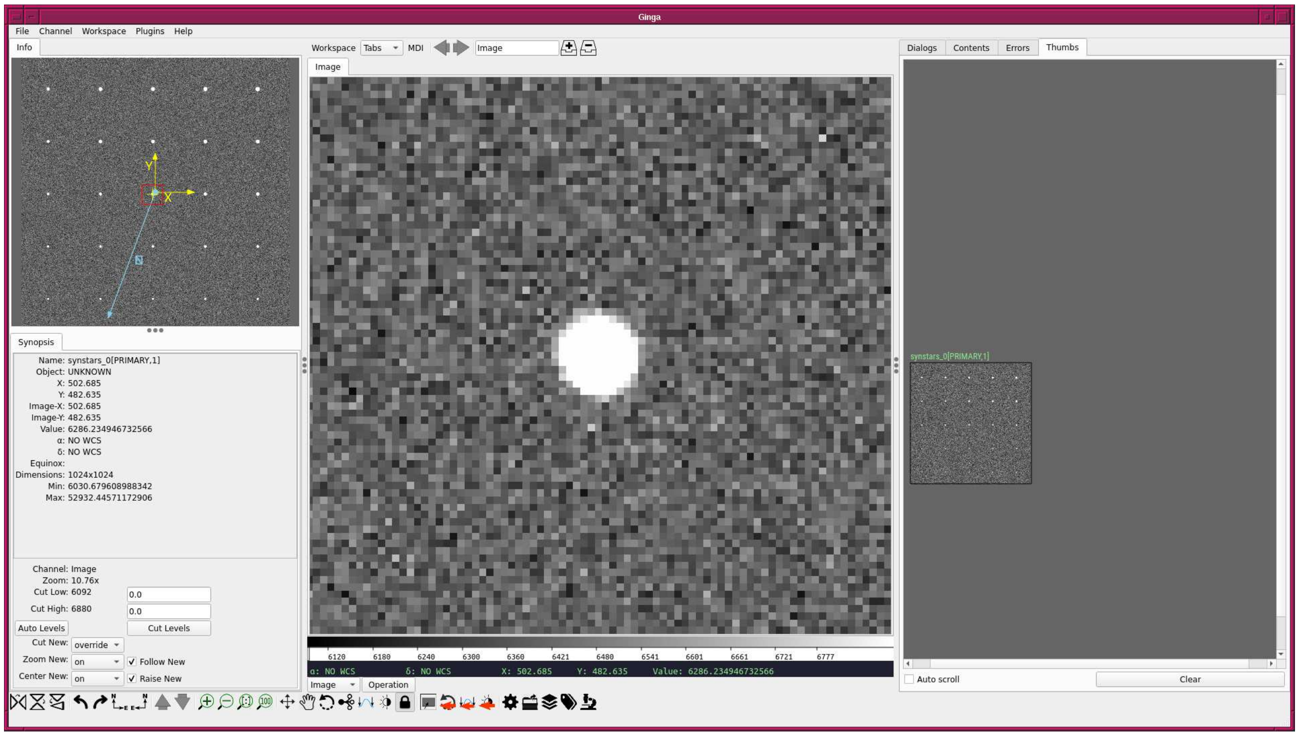
- `https://photutils.readthedocs.io/en/stable/pixel_conventions.html`

3.2 Measuring the centre of the star

Use 2-dimensional Gaussian function to find the centroid and FWHM of the star.

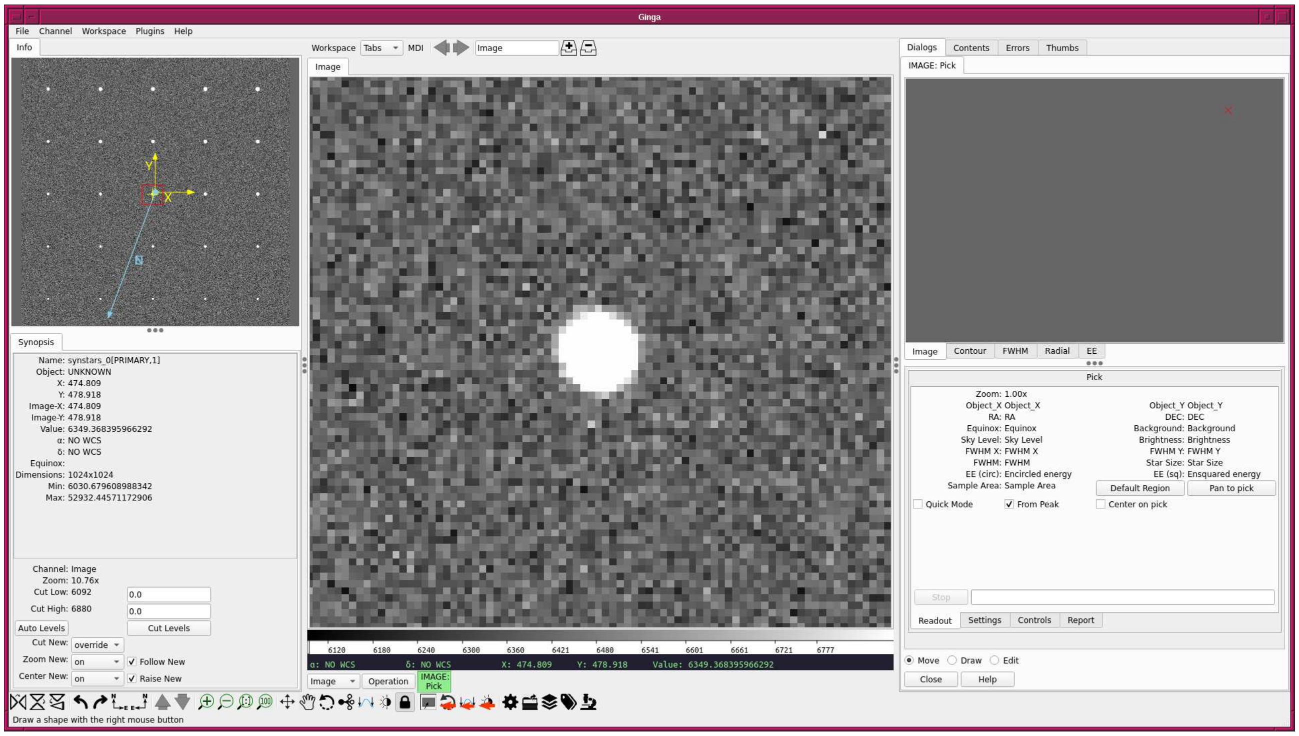
Python Code 3: advobs202202_s11_03.py

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

#
```



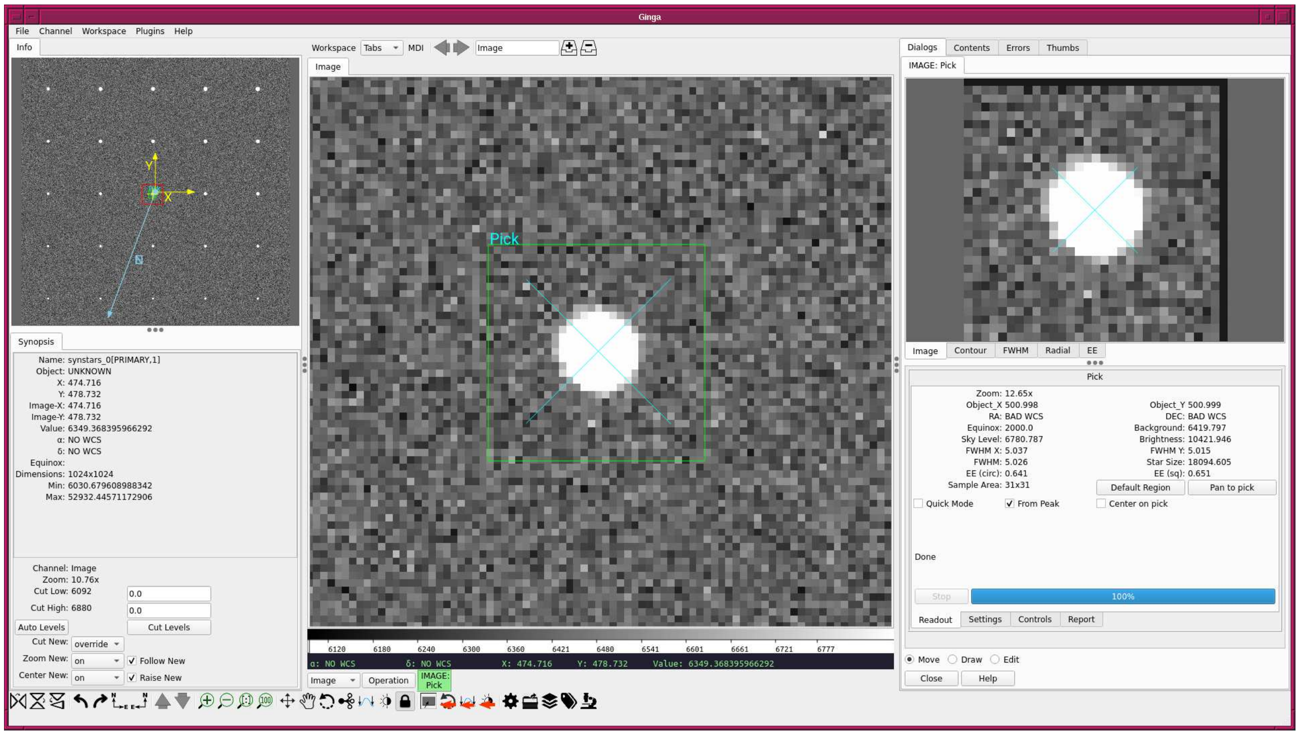
```
       Name: synstars_0[PRIMARY,1]
     Object: UNKNOWN
          X: 502.685
          Y: 482.635
    Image-X: 502.685
    Image-Y: 482.635
      Value: 6286.234946732566
          α: NO WCS
          δ: NO WCS
    Equinox:
 Dimensions: 1024x1024
        Min: 6030.679608988342
        Max: 52932.44571172906

    Channel: Image
       Zoom: 10.76x
    Cut Low: 6092        0.0
   Cut High: 6880        0.0
Auto Levels              Cut Levels
    Cut New: override
   Zoom New: on          ✓ Follow New
 Center New: on          ✓ Raise New

6120 6180 6240 6300 6360 6421 6480 6541 6601 6661 6721 6777
α: NO WCS    δ: NO WCS    X: 502.685    Y: 482.635    Value: 6286.234946732566
```

Figure 5: The target star is shown on the central panel after zooming and panning.


Ginga
File Channel Workspace Plugins Help
Info
Workspace Tabs MDI Image
Image
Dialogs Contents Errors Thumbs
IMAGE: Pick
Synopsis
Image Contour FWHM Radial EE
Image Operation IMAGE: Pick

```
       Name: synstars_0[PRIMARY,1]
     Object: UNKNOWN
          X: 474.809
          Y: 478.918
    Image-X: 474.809
    Image-Y: 478.918
      Value: 6349.368395966292
          α: NO WCS
          δ: NO WCS
    Equinox:
 Dimensions: 1024x1024
        Min: 6030.679608988342
        Max: 52932.44571172906

    Channel: Image
       Zoom: 10.76x
    Cut Low: 6092        0.0
   Cut High: 6880        0.0
Auto Levels              Cut Levels
    Cut New: override
   Zoom New: on          ✓ Follow New
 Center New: on          ✓ Raise New

Draw a shape with the right mouse button

6120 6180 6240 6300 6360 6421 6480 6541 6601 6661 6721 6777
α: NO WCS    δ: NO WCS    X: 474.809    Y: 478.918    Value: 6349.368395966292

                               Pick
        Zoom: 1.00x
    Object_X Object_X                    Object_Y Object_Y
          RA: RA                              DEC: DEC
     Equinox: Equinox                  Background: Background
   Sky Level: Sky Level                Brightness: Brightness
      FWHM X: FWHM X                       FWHM Y: FWHM Y
        FWHM: FWHM                      Star Size: Star Size
   EE (circ): Encircled energy            EE (sq): Ensquared energy
 Sample Area: Sample Area          Default Region   Pan to pick
☐ Quick Mode       ✓ From Peak     ☐ Center on pick

Stop
Readout  Settings  Controls  Report

(●) Move  ( ) Draw  ( ) Edit
Close  Help
```

Figure 6: The function "Pick" is selected.


Ginga
File Channel Workspace Plugins Help
Info
Workspace Tabs MDI Image
Image
Pick
Dialogs Contents Errors Thumbs
IMAGE: Pick
Image Contour FWHM Radial EE
Synopsis
Image Operation IMAGE: Pick

Name: synstars_0[PRIMARY,1]
Object: UNKNOWN
X: 474.716
Y: 478.732
Image-X: 474.716
Image-Y: 478.732
Value: 6349.368395966292
α: NO WCS
δ: NO WCS
Equinox:
Dimensions: 1024x1024
Min: 6030.679608988342
Max: 52932.44571172906

Channel: Image
Zoom: 10.76x
Cut Low: 6092 0.0
Cut High: 6880 0.0
Auto Levels Cut Levels
Cut New: override
Zoom New: on ✓ Follow New
Center New: on ✓ Raise New

α: NO WCS δ: NO WCS X: 474.716 Y: 478.732 Value: 6349.368395966292

Pick

Zoom:	12.65x		
Object_X	500.998	Object_Y	500.999
RA:	BAD WCS	DEC:	BAD WCS
Equinox:	2000.0	Background:	6419.797
Sky Level:	6780.787	Brightness:	10421.946
FWHM X:	5.037	FWHM Y:	5.015
FWHM:	5.026	Star Size:	18094.605
EE (circ):	0.641	EE (sq):	0.651
Sample Area:	31x31		

☐ Quick Mode ✓ From Peak ☐ Center on pick

Done

Stop 100%

Readout Settings Controls Report

● Move ○ Draw ○ Edit

Close Help

Figure 7: The function “Pick” is carried out for the target star, and the results of the measurement is shown in the panel on the right hand side of the window.

```
# Time-stamp: <2022/04/14 12:49:58 (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'
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 and measure the coordinate of the centre of star.

```
% chmod a+x advobs202202_s11_03.py
% ./advobs202202_s11_03.py -h
usage: advobs202202_s11_03.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_s11_03.py -c 2dg -p 2dg -w 15 -x 501 -y 501 -m inferno \
? -o centroid_0.png synstars_0.fits
#
# input parameters
#
#  input file name          = synstars_0.fits
#  half-width of search box = 15.000000
#  x_init                   = 501.000000
#  y_init                   = 501.000000
#  centroid technique       = 2dg
#  output file name         = centroid_0.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  = 500.005626
#   y_centre  = 500.007898
#   amplitude = 10395.434062
#   x_fwhm    = 5.020934
#   y_fwhm    = 5.043127
#   theta     = -21.361006
#
# X_CENTRE, Y_CENTRE, FWHM
500.005626 500.007898 5.032030
# now, generating a plot...
# finished generating a plot!
% ls -lF centroid_0.png
-rw-r--r--  1 daisuke  taiwan  111757 Apr 14 12:52 centroid_0.png
```

The coordinate of the centre of star is measured to be $(x, y) = (500.01, 500.01)$. The FWHM of stellar PSF is measured to be 5.03 pixel.

Show the plot produced to check the result of centroid measurement. (Fig. 8)

```
% feh -dF centroid_0.png
```

4 Aperture photometry

Now, we know the position of a star you selected. We can start aperture photometry to measure the brightness of the star.

4.1 Setting an aperture for the star

First, we set an aperture for the star. For this session, radius of the aperture is set to be twice of FWHM. Make a Python script to set an aperture and make a plot around the star.

Python Code 4: advobs202202_s11_04.py

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

#
# Time-stamp: <2022/04/14 13:33:21 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
```

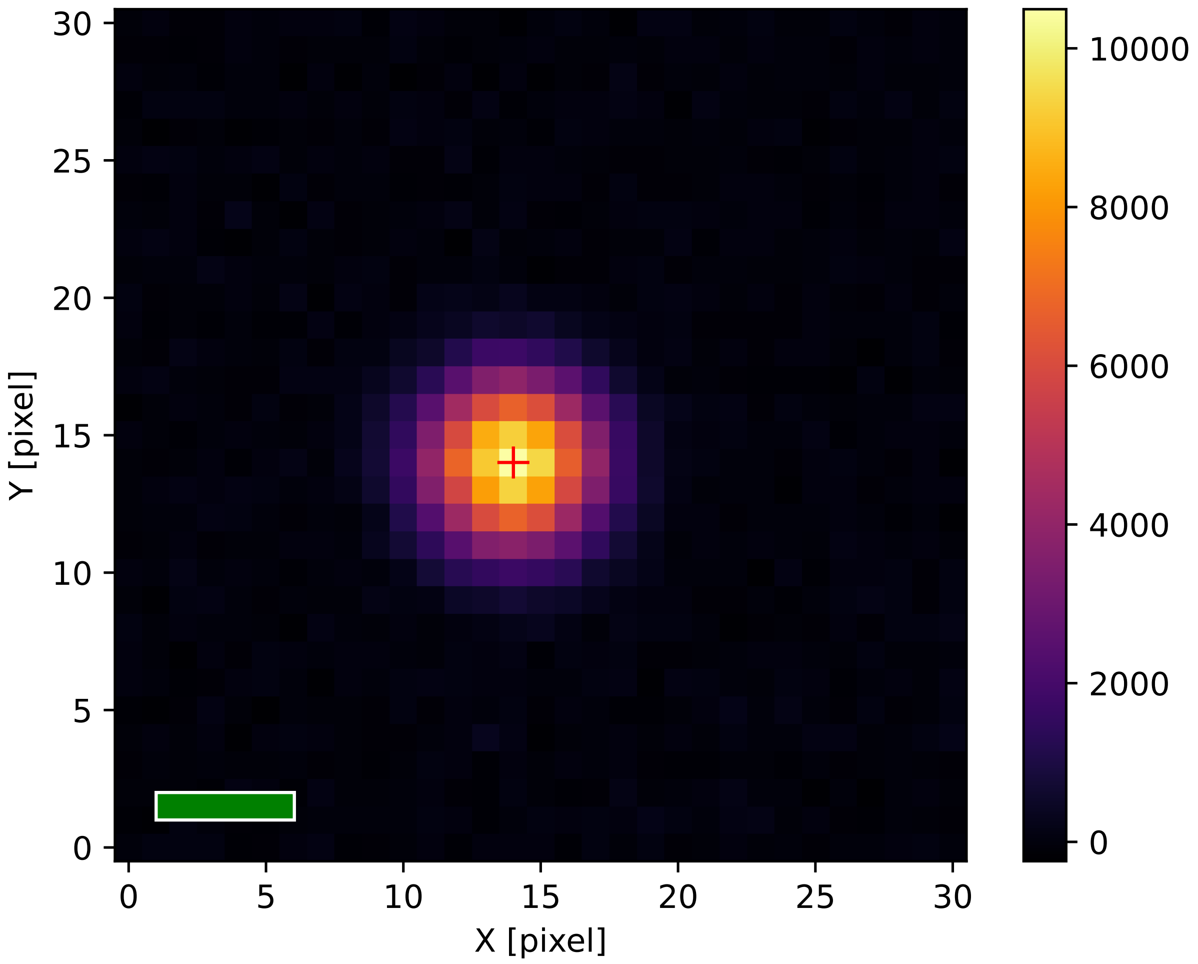


Figure 8: The result of centroid measurement and PSF fitting using 2-dimensional Gaussian function.

```
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
import photutils.aperture

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

# constructing parser object
desc   = 'Setting an aperture'
parser = argparse.ArgumentParser (description=desc)

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

# adding command-line arguments
parser.add_argument ('-f', '--fwhm', type=float, default=4.0, \
                     help='FWHM of stellar PSF in pixel (default: 4.0)')
parser.add_argument ('-a', '--aperture', type=float, default=1.5, \
                     help='aperture radius in FWHM (default: 1.5)')
parser.add_argument ('-w', '--width', type=int, default=15, \
                     help='half-width of subframe to be plotted (default: 15)')
parser.add_argument ('-x', '--xcentre', type=float, default=-1, \
                     help='x coordinate of target')
parser.add_argument ('-y', '--ycentre', type=float, default=-1, \
                     help='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='', \
                     help='output file name')
parser.add_argument ('file', default='', help='input file name')

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

# input parameters
fwhm_pixel            = args.fwhm
aperture_radius_fwhm  = args.aperture
half_width            = args.width
x_centre              = args.xcentre
```

```
y_centre                = args.ycentre
resolution              = args.resolution
cmap                    = args.cmap
file_output             = args.output
file_fits               = args.file

# aperture radius in pixel
aperture_radius_pixel = aperture_radius_fwhm * fwhm_pixel

# 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 status
print ("# now, reading FITS file '%s'..." % file_fits)

# 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_centre and y_centre
    if not ( (x_centre >=0) and (x_centre < image_size_x) ):
        # printing message
        print ("Input x_centre value exceed image size.")
        # exit
        sys.exit ()
    if not ( (y_centre >=0) and (y_centre < image_size_y) ):
        print ("Input y_centre value exceed image size.")
        # printing message
        sys.exit ()
        # exit

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

# printing status
print ("# finished reading FITS file '%s'!" % file_fits)

# printing status
print ("# now, generating an aperture...")

# making subframe
x_min = int (x_centre) - half_width
x_max = int (x_centre) + half_width + 1
y_min = int (y_centre) - half_width
y_max = int (y_centre) + half_width + 1
subframe = data[y_min:y_max, x_min:x_max]

# calculating statistical values
subframe_median = numpy.median (subframe)
subframe_stddev = numpy.std (subframe)

# position of the centre on subframe
x_centre_sub = x_centre - x_min
y_centre_sub = y_centre - y_min
position = (x_centre_sub, y_centre_sub)

# making an aperture
apphot_aperture = photutils.aperture.CircularAperture (position, \
                                                       r=aperture_radius_pixel)

# printing aperture
print (apphot_aperture)

# printing status
print ("# finished generating an aperture!")

# 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, \
                vmin=subframe_median - 3.0 * subframe_stddev, \
                vmax=subframe_median + 6.0 * subframe_stddev)
fig.colorbar (im)
ax.plot (x_centre_sub, y_centre_sub, marker='+', color='blue', markersize=10)

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

# adding a circle to represent aperture for photometry
ap = matplotlib.patches.Circle (xy=position, radius=aperture_radius_pixel, \
                                fill=False, color="yellow", linewidth=3)
ax.add_patch (ap)

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

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

Execute the script.

```
% chmod a+x advobs202202_s11_04.py
% ./advobs202202_s11_04.py -h
usage: advobs202202_s11_04.py [-h] [-f FWHM] [-a APERTURE] [-w WIDTH]
                              [-x XCENTRE] [-y YCENTRE] [-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

Setting an aperture

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -f FWHM, --fwhm FWHM  FWHM of stellar PSF in pixel (default: 4.0)
  -a APERTURE, --aperture APERTURE
                        aperture radius in FWHM (default: 1.5)
  -w WIDTH, --width WIDTH
                        half-width of subframe to be plotted (default: 15)
  -x XCENTRE, --xcentre XCENTRE
                        x coordinate of target
  -y YCENTRE, --ycentre YCENTRE
                        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
```

```
% ./advobs202202_s11_04.py -f 5.03 -a 2.0 -w 20 -x 500.01 -y 500.01 -m inferno \
? -o aperture_0.png synstars_0.fits
# now, reading FITS file 'synstars_0.fits'...
# finished reading FITS file 'synstars_0.fits'!
# now, generating an aperture...
Aperture: CircularAperture
positions: [20.01, 20.01]
r: 10.06
# finished generating an aperture!
# now, generating a plot...
# finished generating a plot!
% ls -lF aperture_0.png
-rw-r--r--  1 daisuke  taiwan  175465 Apr 14 13:36 aperture_0.png
```

Show the plot. (Fig. 9)

```
% feh -dF aperture_0.png
```

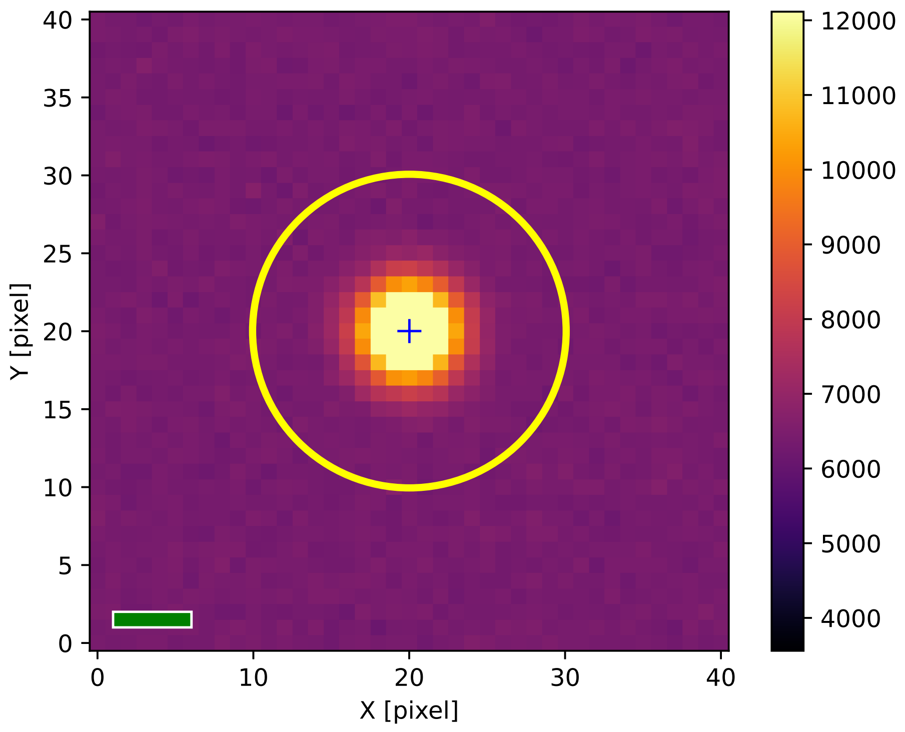


Figure 9: The location of the aperture set for the star. The aperture is shown as yellow circle.

4.2 Setting a sky annulus

Second, set a sky annulus for sky background estimation.

Python Code 5: advobs202202_s11_05.py

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

```
#
# Time-stamp: <2022/04/14 13:54:05 (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
import photutils.aperture

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

# constructing parser object
desc   = 'Setting an aperture and sky annulus'
parser = argparse.ArgumentParser (description=desc)

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

# adding command-line arguments
parser.add_argument ('-f', '--fwhm', type=float, default=4.0, \
                     help='FWHM of stellar PSF in pixel (default: 4.0)')
parser.add_argument ('-a', '--aperture', type=float, default=1.5, \
                     help='aperture radius in FWHM (default: 1.5)')
parser.add_argument ('-s1', '--skyannulus1', type=float, default=3.0, \
                     help='inner sky annulus radius in FWHM (default: 3.0)')
parser.add_argument ('-s2', '--skyannulus2', type=float, default=5.0, \
                     help='outer sky annulus radius in FWHM (default: 5.0)')
parser.add_argument ('-w', '--width', type=int, default=15, \
                     help='half-width of subframe to be plotted (default: 15)')
parser.add_argument ('-x', '--xcentre', type=float, default=-1, \
                     help='x coordinate of target')
parser.add_argument ('-y', '--ycentre', type=float, default=-1, \
                     help='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='', \
                     help='output file name')
parser.add_argument ('file', default='', help='input file name')

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

# input parameters
fwhm_pixel              = args.fwhm
aperture_radius_fwhm    = args.aperture
skyannulus_inner_fwhm = args.skyannulus1
skyannulus_outer_fwhm = args.skyannulus2
half_width              = args.width
x_centre                = args.xcentre
y_centre                = args.ycentre
resolution              = args.resolution
cmap                    = args.cmap
file_output             = args.output
file_fits               = args.file

# aperture radius and sky annulus in pixel
aperture_radius_pixel  = aperture_radius_fwhm * fwhm_pixel
skyannulus_inner_pixel = skyannulus_inner_fwhm * fwhm_pixel
skyannulus_outer_pixel = skyannulus_outer_fwhm * fwhm_pixel

# 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 status
print ("# now, reading FITS file '%s'..." % file_fits)

# 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_centre and y_centre
    if not ( (x_centre >=0) and (x_centre < image_size_x) ):
        # printing message
        print ("Input x_centre value exceed image size.")
        # exit
        sys.exit ()
    if not ( (y_centre >=0) and (y_centre < image_size_y) ):
        print ("Input y_centre value exceed image size.")
        # printing message
        sys.exit ()
        # exit

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

# printing status
print ("# finished reading FITS file '%s'!" % file_fits)

# printing status
print ("# now, generating an aperture...")

# making subframe
x_min = int (x_centre) - half_width
x_max = int (x_centre) + half_width + 1
y_min = int (y_centre) - half_width
y_max = int (y_centre) + half_width + 1
subframe = data[y_min:y_max, x_min:x_max]

# calculating statistical values
subframe_median = numpy.median (subframe)
subframe_stddev = numpy.std (subframe)

# position of the centre on subframe
x_centre_sub = x_centre - x_min
y_centre_sub = y_centre - y_min
position = (x_centre_sub, y_centre_sub)

# making an aperture
apphot_aperture = photutils.aperture.CircularAperture (position, \
                                                       r=aperture_radius_pixel)

# making a sky annulus
apphot_annulus \
    = photutils.aperture.CircularAnnulus (position, \
                                          r_in=skyannulus_inner_pixel, \
                                          r_out=skyannulus_outer_pixel)

# printing aperture
print ("aperture for star:")
print (apphot_aperture)

# printing sky annulus
print ("sky annulus:")
print (apphot_annulus)

# printing status
print ("# finished generating an aperture!")

# 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, \
                vmin=subframe_median - 3.0 * subframe_stddev, \
                vmax=subframe_median + 6.0 * subframe_stddev)
fig.colorbar (im)
ax.plot (x_centre_sub, y_centre_sub, marker='+', color='blue', markersize=10)

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

# adding a circle to represent aperture for photometry
ap = matplotlib.patches.Circle (xy=position, radius=aperture_radius_pixel, \
                                fill=False, color="yellow", linewidth=3)
ax.add_patch (ap)

# adding circles to represent sky annulus for photometry
annulus_i = matplotlib.patches.Circle (xy=position, \
                                       radius=skyannulus_inner_pixel, \
                                       fill=False, color="cyan", \
                                       linewidth=3, linestyle='--')
annulus_o = matplotlib.patches.Circle (xy=position, \
                                       radius=skyannulus_outer_pixel, \
                                       fill=False, color="cyan", \
                                       linewidth=3, linestyle='--')
ax.add_patch (annulus_i)
ax.add_patch (annulus_o)

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

# printing status
```

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

Run the script. For here, we set the inner radius of sky annulus to be 3 times of FWHM and the outer radius of sky annulus to be 6 times of FWHM.

```
% chmod a+x advobs202202_s11_05.py
% ./advobs202202_s11_05.py -h
usage: advobs202202_s11_05.py [-h] [-f FWHM] [-a APERTURE] [-s1 SKYANNULUS1]
                              [-s2 SKYANNULUS2] [-w WIDTH] [-x XCENTRE]
                              [-y YCENTRE] [-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

Setting an aperture

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -f FWHM, --fwhm FWHM  FWHM of stellar PSF in pixel (default: 4.0)
  -a APERTURE, --aperture APERTURE
                        aperture radius in FWHM (default: 1.5)
  -s1 SKYANNULUS1, --skyannulus1 SKYANNULUS1
                        inner sky annulus radius in FWHM (default: 3.0)
  -s2 SKYANNULUS2, --skyannulus2 SKYANNULUS2
                        outer sky annulus radius in FWHM (default: 5.0)
  -w WIDTH, --width WIDTH
                        half-width of subframe to be plotted (default: 15)
  -x XCENTRE, --xcentre XCENTRE
                        x coordinate of target
  -y YCENTRE, --ycentre YCENTRE
                        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


% chmod a+x ao2021_s11_05.py
% ./ao2021_s11_05.py -h
usage: ao2021_s11_05.py [-h] [-f FWHM] [-a APERTURE] [-s1 SKYANNULUS1]
                        [-s2 SKYANNULUS2] [-w WIDTH] [-x XCENTRE] [-y YCENTRE]
                        [-o OUTPUT]
                        file

Setting an aperture

positional arguments:
  file                  input file name
```

```
optional arguments:
  -h, --help            show this help message and exit
  -f FWHM, --fwhm FWHM  FWHM of stellar PSF in pixel (default: 4.0)
  -a APERTURE, --aperture APERTURE
                        aperture radius in FWHM (default: 1.5)
  -s1 SKYANNULUS1, --skyannulus1 SKYANNULUS1
                        inner sky annulus radius in FWHM (default: 3.0)
  -s2 SKYANNULUS2, --skyannulus2 SKYANNULUS2
                        outer sky annulus radius in FWHM (default: 5.0)
  -w WIDTH, --width WIDTH
                        half-width of subframe to be plotted (default: 15)
  -x XCENTRE, --xcentre XCENTRE
                        x coordinate of target
  -y YCENTRE, --ycentre YCENTRE
                        y coordinate of target
  -o OUTPUT, --output OUTPUT
                        output file name

% ./advobs202202_s11_05.py -f 5.03 -a 2.0 -s1 3.0 -s2 6.0 -w 35 \
? -x 500.01 -y 500.01 -m inferno -o aperture_1.png synstars_0.fits
# now, reading FITS file 'synstars_0.fits'...
# finished reading FITS file 'synstars_0.fits'!
# now, generating an aperture...
aperture for star:
Aperture: CircularAperture
positions: [35.01, 35.01]
r: 10.06
sky annulus:
Aperture: CircularAnnulus
positions: [35.01, 35.01]
r_in: 15.09
r_out: 30.18
# finished generating an aperture!
# now, generating a plot...
# finished generating a plot!
% ls -lF aperture_1.png
-rw-r--r--  1 daisuke  taiwan  247157 Apr 14 13:50 aperture_1.png
```

Show the plot to visualise the locations of aperture and sky annulus. (Fig. 10)

4.3 Adding pixel values within the aperture

Third, we add pixel values within the aperture. The function `aperture_photometry ()` can be used for this.

Python Code 6: advobs202202_s11_06.py

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

#
# Time-stamp: <2022/04/14 15:03:01 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys
```

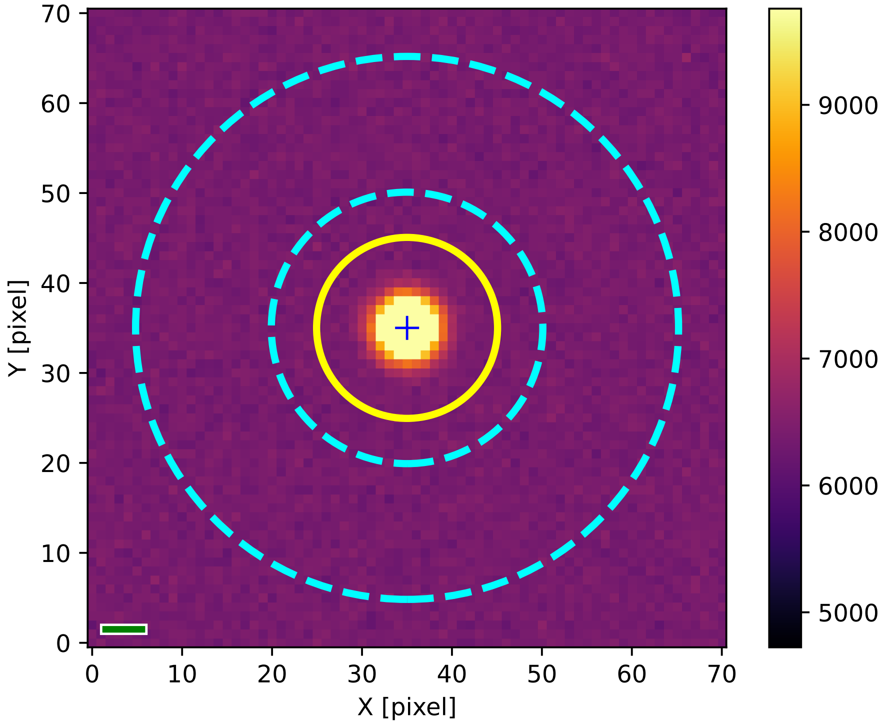


Figure 10: The location of the aperture and sky annulus set for the star. The circular aperture for the star is shown by yellow circle, and the circular sky annulus for sky background estimate is shown by cyan dashed circle.

```
# 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
import photutils.aperture

# constructing parser object
desc   = 'adding all the signals within aperture'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-f', '--fwhm', type=float, default=4.0, \
                     help='FWHM of stellar PSF in pixel (default: 4.0)')
parser.add_argument ('-a', '--aperture', type=float, default=1.5, \
                     help='aperture radius in FWHM (default: 1.5)')
parser.add_argument ('-s1', '--skyannulus1', type=float, default=3.0, \
                     help='inner sky annulus radius in FWHM (default: 3.0)')
parser.add_argument ('-s2', '--skyannulus2', type=float, default=5.0, \
                     help='outer sky annulus radius in FWHM (default: 5.0)')
parser.add_argument ('-w', '--width', type=int, default=15, \
                     help='half-width of subframe to be plotted (default: 15)')
parser.add_argument ('-x', '--xcentre', type=float, default=-1, \
                     help='x coordinate of target')
parser.add_argument ('-y', '--ycentre', type=float, default=-1, \
                     help='y coordinate of target')
parser.add_argument ('file', default='', help='input file name')

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

# input parameters
fwhm_pixel             = args.fwhm
aperture_radius_fwhm   = args.aperture
skyannulus_inner_fwhm  = args.skyannulus1
skyannulus_outer_fwhm  = args.skyannulus2
half_width             = args.width
x_centre               = args.xcentre
y_centre               = args.ycentre
file_fits              = args.file

# aperture radius and sky annulus in pixel
aperture_radius_pixel   = aperture_radius_fwhm * fwhm_pixel
skyannulus_inner_pixel  = skyannulus_inner_fwhm * fwhm_pixel
skyannulus_outer_pixel  = skyannulus_outer_fwhm * fwhm_pixel

# making pathlib objects
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 status
print ("# now, reading FITS file '%s'..." % file_fits)

# 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_centre and y_centre
    if not ( (x_centre >=0) and (x_centre < image_size_x) ):
        # printing message
        print ("Input x_centre value exceed image size.")
        # exit
        sys.exit ()
    if not ( (y_centre >=0) and (y_centre < image_size_y) ):
        print ("Input y_centre value exceed image size.")
        # printing message
        sys.exit ()
        # exit

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

# printing status
print ("# finished reading FITS file '%s'!" % file_fits)

# printing status
print ("# now, generating an aperture...")

# making subframe
x_min = int (x_centre) - half_width
x_max = int (x_centre) + half_width + 1
y_min = int (y_centre) - half_width
y_max = int (y_centre) + half_width + 1
subframe = data[y_min:y_max, x_min:x_max]

# calculating statistical values
subframe_median = numpy.median (subframe)
subframe_stddev = numpy.std (subframe)
```

```
# position of the centre on subframe
x_centre_sub = x_centre - x_min
y_centre_sub = y_centre - y_min
position = (x_centre_sub, y_centre_sub)

# making an aperture
apphot_aperture = photutils.aperture.CircularAperture (position, \
                                                       r=aperture_radius_pixel)

# making a sky annulus
apphot_annulus \
    = photutils.aperture.CircularAnnulus (position, \
                                          r_in=skyannulus_inner_pixel, \
                                          r_out=skyannulus_outer_pixel)

# printing aperture
print ("aperture for star:")
print (apphot_aperture)

# printing sky annulus
print ("sky annulus:")
print (apphot_annulus)

# printing status
print ("# finished generating an aperture!")

# printing status
print ("# now, adding all the signal values within aperture...")

# adding all the signal values within the aperture
apphot_star \
    = photutils.aperture.aperture_photometry (subframe, apphot_aperture, \
                                              error=numpy.sqrt (subframe))

# printing result
print (apphot_star)
print ("aperture sum       = %f ADU" % apphot_star['aperture_sum'])
print ("aperture sum error = %f ADU" % apphot_star['aperture_sum_err'])

# printing status
print ("# finished adding all the signal values within aperture!")
```

Run the script.

```
% chmod a+x advobs202202_s11_06.py
% ./advobs202202_s11_06.py -h
usage: advobs202202_s11_06.py [-h] [-f FWHM] [-a APERTURE] [-s1 SKYANNULUS1]
                              [-s2 SKYANNULUS2] [-w WIDTH] [-x XCENTRE]
                              [-y YCENTRE]
                              file

adding all the signals within aperture

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -f FWHM, --fwhm FWHM  FWHM of stellar PSF in pixel (default: 4.0)
  -a APERTURE, --aperture APERTURE
                        aperture radius in FWHM (default: 1.5)
  -s1 SKYANNULUS1, --skyannulus1 SKYANNULUS1
                        inner sky annulus radius in FWHM (default: 3.0)
  -s2 SKYANNULUS2, --skyannulus2 SKYANNULUS2
                        outer sky annulus radius in FWHM (default: 5.0)
  -w WIDTH, --width WIDTH
                        half-width of subframe to be plotted (default: 15)
  -x XCENTRE, --xcentre XCENTRE
                        x coordinate of target
  -y YCENTRE, --ycentre YCENTRE
                        y coordinate of target

% ./advobs202202_s11_06.py -f 5.03 -a 2.0 -s1 3.0 -s2 6.0 -w 35 \
? -x 500.01 -y 500.01 synstars_0.fits
# now, reading FITS file 'synstars_0.fits'...
# finished reading FITS file 'synstars_0.fits'!
# now, generating an aperture...
aperture for star:
Aperture: CircularAperture
positions: [35.01, 35.01]
r: 10.06
sky annulus:
Aperture: CircularAnnulus
positions: [35.01, 35.01]
r_in: 15.09
r_out: 30.18
# finished generating an aperture!
# now, adding all the signal values within aperture...
 id      xcenter           ycenter          aperture_sum    aperture_sum_err
           pix               pix
--- ----------------- ----------------- ----------------- ------------------
  1 35.00999999999999 35.00999999999999 2335031.584554946 1528.0810137407461
aperture sum       = 2335031.584555 ADU
aperture sum error = 1528.081014 ADU
# finished adding all the signal values within aperture!
```

The total flux within the aperture is measured to be 2.34×10^6 ADU. Note that the flux we have measured is the flux of combined signal of the star and sky background. To know the net flux of the star, we have to subtract sky background component.

4.4 A simple way to estimate sky background level

A very simple way to estimate sky background level is to calculate the mean value of pixels within the sky annulus. Use sigma-clipping algorithm to calculate the mean value. Here is a Python script to do it.

Python Code 7: advobs202202_s11_07.py

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

#
# Time-stamp: <2022/04/14 15:02:26 (CST) daisuke>
#

# importing argparse module
import argparse

```

```
# importing sys module
import sys

# importing pathlib module
import pathlib

# importing numpy module
import numpy

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

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

# constructing parser object
desc   = 'estimating sky background level'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-f', '--fwhm', type=float, default=4.0, \
                     help='FWHM of stellar PSF in pixel (default: 4.0)')
parser.add_argument ('-a', '--aperture', type=float, default=1.5, \
                     help='aperture radius in FWHM (default: 1.5)')
parser.add_argument ('-s1', '--skyannulus1', type=float, default=3.0, \
                     help='inner sky annulus radius in FWHM (default: 3.0)')
parser.add_argument ('-s2', '--skyannulus2', type=float, default=5.0, \
                     help='outer sky annulus radius in FWHM (default: 5.0)')
parser.add_argument ('-w', '--width', type=int, default=15, \
                     help='half-width of subframe to be plotted (default: 15)')
parser.add_argument ('-x', '--xcentre', type=float, default=-1, \
                     help='x coordinate of target')
parser.add_argument ('-y', '--ycentre', type=float, default=-1, \
                     help='y coordinate of target')
parser.add_argument ('file', default='', help='input file name')

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

# input parameters
fwhm_pixel             = args.fwhm
aperture_radius_fwhm   = args.aperture
skyannulus_inner_fwhm = args.skyannulus1
skyannulus_outer_fwhm = args.skyannulus2
half_width             = args.width
x_centre               = args.xcentre
y_centre               = args.ycentre
file_fits              = args.file

# aperture radius and sky annulus in pixel
aperture_radius_pixel   = aperture_radius_fwhm * fwhm_pixel
skyannulus_inner_pixel = skyannulus_inner_fwhm * fwhm_pixel
skyannulus_outer_pixel = skyannulus_outer_fwhm * fwhm_pixel

# making pathlib objects
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 status
print ("# now, reading FITS file '%s'..." % file_fits)

# 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_centre and y_centre
    if not ( (x_centre >=0) and (x_centre < image_size_x) ):
        # printing message
        print ("Input x_centre value exceed image size.")
        # exit
        sys.exit ()
    if not ( (y_centre >=0) and (y_centre < image_size_y) ):
        print ("Input y_centre value exceed image size.")
        # printing message
        sys.exit ()
        # exit

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

# printing status
print ("# finished reading FITS file '%s'!" % file_fits)

# printing status
print ("# now, generating an aperture...")

# making subframe
x_min = int (x_centre) - half_width
x_max = int (x_centre) + half_width + 1
y_min = int (y_centre) - half_width
y_max = int (y_centre) + half_width + 1
subframe = data[y_min:y_max, x_min:x_max]
```

```
# calculating statistical values
subframe_median = numpy.median (subframe)
subframe_stddev = numpy.std (subframe)

# position of the centre on subframe
x_centre_sub = x_centre - x_min
y_centre_sub = y_centre - y_min
position = (x_centre_sub, y_centre_sub)

# making an aperture
apphot_aperture = photutils.aperture.CircularAperture (position, \
                                                       r=aperture_radius_pixel)

# making a sky annulus
apphot_annulus \
    = photutils.aperture.CircularAnnulus (position, \
                                          r_in=skyannulus_inner_pixel, \
                                          r_out=skyannulus_outer_pixel)

# printing aperture
print ("aperture for star:")
print (apphot_aperture)

# printing sky annulus
print ("sky annulus:")
print (apphot_annulus)

# printing status
print ("# finished generating an aperture!")

# printing status
print ("# now, adding all the signal values within aperture...")

# adding all the signal values within the aperture
apphot_star \
    = photutils.aperture.aperture_photometry (subframe, apphot_aperture, \
                                              error=numpy.sqrt (subframe))

# printing result
print (apphot_star)
print ("aperture sum       = %f ADU" % apphot_star['aperture_sum'])
print ("aperture sum error = %f ADU" % apphot_star['aperture_sum_err'])

# printing status
print ("# finished adding all the signal values within aperture!")

# printing status
print ("# now, estimating sky background value...")

# sky background estimate
sigma_clip_4 = astropy.stats.SigmaClip (sigma=4.0, maxiters=10)
apphot_sky_stats \
    = photutils.aperture.ApertureStats (subframe, apphot_annulus, \
                                        sigma_clip=sigma_clip_4)
skybg_per_pixel     = apphot_sky_stats.mean
skybg_err_per_pixel = apphot_sky_stats.std

# printing result
```

```
print ("sky background       = %f ADU/pix" % skybg_per_pixel)
print ("sky background error = %f ADU/pix" % skybg_err_per_pixel)

# printing status
print ("# finished estimating sky background value!")
```

Execute the script and measure the sky background level.

```
% chmod a+x advobs202202_s11_07.py
% ./advobs202202_s11_07.py -h
usage: advobs202202_s11_07.py [-h] [-f FWHM] [-a APERTURE] [-s1 SKYANNULUS1]
                              [-s2 SKYANNULUS2] [-w WIDTH] [-x XCENTRE]
                              [-y YCENTRE]
                              file

adding all the signals within aperture

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -f FWHM, --fwhm FWHM  FWHM of stellar PSF in pixel (default: 4.0)
  -a APERTURE, --aperture APERTURE
                        aperture radius in FWHM (default: 1.5)
  -s1 SKYANNULUS1, --skyannulus1 SKYANNULUS1
                        inner sky annulus radius in FWHM (default: 3.0)
  -s2 SKYANNULUS2, --skyannulus2 SKYANNULUS2
                        outer sky annulus radius in FWHM (default: 5.0)
  -w WIDTH, --width WIDTH
                        half-width of subframe to be plotted (default: 15)
  -x XCENTRE, --xcentre XCENTRE
                        x coordinate of target
  -y YCENTRE, --ycentre YCENTRE
                        y coordinate of target

% ./advobs202202_s11_07.py -f 5.03 -a 2.0 -s1 3.0 -s2 6.0 -w 35 \
? -x 500.01 -y 500.01 synstars_0.fits
# now, reading FITS file 'synstars_0.fits'...
# finished reading FITS file 'synstars_0.fits'!
# now, generating an aperture...
aperture for star:
Aperture: CircularAperture
positions: [35.01, 35.01]
r: 10.06
sky annulus:
Aperture: CircularAnnulus
positions: [35.01, 35.01]
r_in: 15.09
r_out: 30.18
# finished generating an aperture!
# now, adding all the signal values within aperture...
 id      xcenter           ycenter         aperture_sum    aperture_sum_err
           pix               pix
--- ----------------- ----------------- ----------------- ------------------
  1 35.00999999999999 35.00999999999999 2335031.584554946 1528.0810137407461
aperture sum       = 2335031.584555 ADU
aperture sum error = 1528.081014 ADU
```

```
# finished adding all the signal values within aperture!
# now, estimating sky background value...
sky background       = 6400.339872 ADU/pix
sky background error = 79.350522 ADU/pix
# finished estimating sky background value!
```

Measured sky background level is 6400.3 ± 79.4 ADU which is consistent with the input parameter given to the Python script for synthetic image generation.

4.5 Result of sky background subtraction

Subtract the sky background from the sum of pixel values within the aperture.

Python Code 8: advobs202202_s11_08.py

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

#
# Time-stamp: <2022/04/22 19:24:04 (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
import astropy.stats

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

# constructing parser object
desc   = 'calculating net flux of star'
parser = argparse.ArgumentParser (description=desc)

# adding command-line arguments
parser.add_argument ('-f', '--fwhm', type=float, default=4.0, \
                     help='FWHM of stellar PSF in pixel (default: 4.0)')
parser.add_argument ('-a', '--aperture', type=float, default=1.5, \
                     help='aperture radius in FWHM (default: 1.5)')
parser.add_argument ('-s1', '--skyannulus1', type=float, default=3.0, \
                     help='inner sky annulus radius in FWHM (default: 3.0)')
parser.add_argument ('-s2', '--skyannulus2', type=float, default=5.0, \
                     help='outer sky annulus radius in FWHM (default: 5.0)')
parser.add_argument ('-w', '--width', type=int, default=15, \
                     help='half-width of subframe to be plotted (default: 15)')
parser.add_argument ('-x', '--xcentre', type=float, default=-1, \
                     help='x coordinate of target')
```

```
parser.add_argument ('-y', '--ycentre', type=float, default=-1, \
                     help='y coordinate of target')
parser.add_argument ('file', default='', help='input file name')

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

# input parameters
fwhm_pixel             = args.fwhm
aperture_radius_fwhm   = args.aperture
skyannulus_inner_fwhm  = args.skyannulus1
skyannulus_outer_fwhm  = args.skyannulus2
half_width             = args.width
x_centre               = args.xcentre
y_centre               = args.ycentre
file_fits              = args.file

# aperture radius and sky annulus in pixel
aperture_radius_pixel   = aperture_radius_fwhm * fwhm_pixel
skyannulus_inner_pixel  = skyannulus_inner_fwhm * fwhm_pixel
skyannulus_outer_pixel  = skyannulus_outer_fwhm * fwhm_pixel

# making pathlib objects
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 status
print ("# now, reading FITS file '%s'..." % file_fits)

# 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_centre and y_centre
    if not ( (x_centre >=0) and (x_centre < image_size_x) ):
        # printing message
        print ("Input x_centre value exceed image size.")
        # exit
        sys.exit ()
    if not ( (y_centre >=0) and (y_centre < image_size_y) ):
        print ("Input y_centre value exceed image size.")
        # printing message
        sys.exit ()
        # exit

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

# printing status
print ("# finished reading FITS file '%s'!" % file_fits)

# printing status
print ("# now, generating an aperture...")

# making subframe
x_min = int (x_centre) - half_width
x_max = int (x_centre) + half_width + 1
y_min = int (y_centre) - half_width
y_max = int (y_centre) + half_width + 1
subframe = data[y_min:y_max, x_min:x_max]

# calculating statistical values
subframe_median = numpy.median (subframe)
subframe_stddev = numpy.std (subframe)

# position of the centre on subframe
x_centre_sub = x_centre - x_min
y_centre_sub = y_centre - y_min
position = (x_centre_sub, y_centre_sub)

# making an aperture
apphot_aperture = photutils.aperture.CircularAperture (position, \
                                                       r=aperture_radius_pixel)

# making a sky annulus
apphot_annulus \
    = photutils.aperture.CircularAnnulus (position, \
                                          r_in=skyannulus_inner_pixel, \
                                          r_out=skyannulus_outer_pixel)

# printing aperture
print ("aperture for star:")
print (apphot_aperture)

# printing sky annulus
print ("sky annulus:")
print (apphot_annulus)

# printing status
print ("# finished generating an aperture!")

# printing status
print ("# now, adding all the signal values within aperture...")

# adding all the signal values within the aperture
```

```
apphot_star = photutils.aperture.aperture_photometry (subframe, apphot_aperture)

# raw flux = star + sky
raw_flux    = apphot_star['aperture_sum']

# printing result
print (apphot_star)
print ("aperture sum          = %f ADU" % raw_flux)

# printing status
print ("# finished adding all the signal values within aperture!")

# printing status
print ("# now, estimating sky background value...")

# sky background estimate
sigma_clip_4 = astropy.stats.SigmaClip (sigma=4.0, maxiters=10)
apphot_sky_stats \
    = photutils.aperture.ApertureStats (subframe, apphot_annulus, \
                                        sigma_clip=sigma_clip_4)
skybg_per_pixel     = apphot_sky_stats.mean
skybg_err_per_pixel = apphot_sky_stats.std

# printing result
print ("sky background       = %f ADU/pix" % skybg_per_pixel)
print ("sky background error = %f ADU/pix" % skybg_err_per_pixel)

# printing status
print ("# finished estimating sky background value!")

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

# sky background subtraction

# net flux = (total flux within aperture)
#              - (skybg per pixel) * (number of pixels in aperture)
npix         = apphot_aperture.area
net_flux     = raw_flux - skybg_per_pixel * npix
net_flux_err = numpy.sqrt (raw_flux + npix * skybg_err_per_pixel**2)

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

# printing result of aperture photometry
print ("#")
print ("# result of aperture photometry")
print ("#")
print ("# X, Y, APERTURE_RADIUS, OUTER_SKY_ANNULUS, INNER_SKY_ANNULUS,")
print ("# NPIX_APERTURE, FLUX, SKY_PER_PIX, SKY_ERR_PER_PIX,")
print ("# NET_FLUX, NET_FLUX_ERR")
print ("#")
print ("%f %f %f %f %f %f %f %f %f %f %f" \
       % (x_centre, y_centre, aperture_radius_pixel, \
          skyannulus_inner_pixel, skyannulus_outer_pixel, npix, \
          raw_flux, skybg_per_pixel, skybg_err_per_pixel, \
          net_flux, net_flux_err) )
```

Run the script, and check the result.

```
% chmod a+x advobs202202_s11_08.py
% ./advobs202202_s11_08.py -h
usage: advobs202202_s11_08.py [-h] [-f FWHM] [-a APERTURE] [-s1 SKYANNULUS1]
                              [-s2 SKYANNULUS2] [-w WIDTH] [-x XCENTRE]
                              [-y YCENTRE]
                              file

calculating net flux of star

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -f FWHM, --fwhm FWHM  FWHM of stellar PSF in pixel (default: 4.0)
  -a APERTURE, --aperture APERTURE
                        aperture radius in FWHM (default: 1.5)
  -s1 SKYANNULUS1, --skyannulus1 SKYANNULUS1
                        inner sky annulus radius in FWHM (default: 3.0)
  -s2 SKYANNULUS2, --skyannulus2 SKYANNULUS2
                        outer sky annulus radius in FWHM (default: 5.0)
  -w WIDTH, --width WIDTH
                        half-width of subframe to be plotted (default: 15)
  -x XCENTRE, --xcentre XCENTRE
                        x coordinate of target
  -y YCENTRE, --ycentre YCENTRE
                        y coordinate of target

% ./advobs202202_s11_08.py -f 5.03 -a 2.0 -s1 3.0 -s2 6.0 -w 35 \
? -x 500.01 -y 500.01 synstars_0.fits
# now, reading FITS file 'synstars_0.fits'...
# finished reading FITS file 'synstars_0.fits'!
# now, generating an aperture...
aperture for star:
Aperture: CircularAperture
positions: [35.01, 35.01]
r: 10.06
sky annulus:
Aperture: CircularAnnulus
positions: [35.01, 35.01]
r_in: 15.09
r_out: 30.18
# finished generating an aperture!
# now, adding all the signal values within aperture...
 id      xcenter           ycenter          aperture_sum
           pix               pix
--- ----------------- ----------------- -----------------
  1 35.00999999999999 35.00999999999999 2335031.584554946
aperture sum         = 2335031.584555 ADU
# finished adding all the signal values within aperture!
# now, estimating sky background value...
sky background       = 6400.339872 ADU/pix
sky background error = 79.350522 ADU/pix
# finished estimating sky background value!
# now, subtracting sky background...
# finished subtracting sky background!
#
# result of aperture photometry
```

```
#
# X, Y, APERTURE_RADIUS, OUTER_SKY_ANNULUS, INNER_SKY_ANNULUS,
# NPIX_APERTURE, FLUX, SKY_PER_PIX, SKY_ERR_PER_PIX,
# NET_FLUX, NET_FLUX_ERR
#
500.010000 500.010000 10.060000 15.090000 30.180000 317.940486 2335031.584555 64
00.339872 79.350522 300104.413288 2082.533443
```

Estimated net flux of the star is 300,100 ± 2,083 ADU and it is consistent with what we expect (300,000 ADU).

4.6 Extraction of pixel values within sky annulus

Make a Python script to extract pixel values within sky annulus.

Python Code 9: advobs202202_s11_09.py

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

#
# Time-stamp: <2022/04/14 15:45:36 (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
import photutils.aperture

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

# constructing parser object
desc   = 'extraction of pixel values within sky annulus'
parser = argparse.ArgumentParser (description=desc)

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

# adding command-line arguments
```

```
parser.add_argument ('-f', '--fwhm', type=float, default=4.0, \
                     help='FWHM of stellar PSF in pixel (default: 4.0)')
parser.add_argument ('-a', '--aperture', type=float, default=1.5, \
                     help='aperture radius in FWHM (default: 1.5)')
parser.add_argument ('-s1', '--skyannulus1', type=float, default=3.0, \
                     help='inner sky annulus radius in FWHM (default: 3.0)')
parser.add_argument ('-s2', '--skyannulus2', type=float, default=5.0, \
                     help='outer sky annulus radius in FWHM (default: 5.0)')
parser.add_argument ('-w', '--width', type=int, default=15, \
                     help='half-width of subframe to be plotted (default: 15)')
parser.add_argument ('-x', '--xcentre', type=float, default=-1, \
                     help='x coordinate of target')
parser.add_argument ('-y', '--ycentre', type=float, default=-1, \
                     help='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='', \
                     help='output file name')
parser.add_argument ('file', nargs=1, default='', help='input file name')

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

# input parameters
fwhm_pixel              = args.fwhm
aperture_radius_fwhm    = args.aperture
skyannulus_inner_fwhm = args.skyannulus1
skyannulus_outer_fwhm = args.skyannulus2
half_width              = args.width
x_centre                = args.xcentre
y_centre                = args.ycentre
resolution              = args.resolution
cmap                    = args.cmap
file_output             = args.output
file_fits               = args.file[0]

# aperture radii in pixel
aperture_radius_pixel  = aperture_radius_fwhm * fwhm_pixel
skyannulus_inner_pixel = skyannulus_inner_fwhm * fwhm_pixel
skyannulus_outer_pixel = skyannulus_outer_fwhm * fwhm_pixel

# 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 ()

# 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_centre and y_centre
    if not ( (x_centre >=0) and (x_centre < image_size_x) ):
        # printing message
        print ("Input x_centre value exceed image size.")
        # exit
        sys.exit ()
    if not ( (y_centre >=0) and (y_centre < image_size_y) ):
        print ("Input y_centre value exceed image size.")
        # printing message
        sys.exit ()
        # exit

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

# making subframe
x_min = int (x_centre) - half_width
x_max = int (x_centre) + half_width + 1
y_min = int (y_centre) - half_width
y_max = int (y_centre) + half_width + 1

# position of the centre
position = (x_centre, y_centre)

# making apertures (circular aperture for star and circular annulus for sky)
apphot_aperture \
    = photutils.aperture.CircularAperture (position, r=aperture_radius_pixel)
apphot_annulus \
    = photutils.aperture.CircularAnnulus (position, \
                                          r_in=skyannulus_inner_pixel, \
                                               r_out=skyannulus_outer_pixel)

# making masked data for sky annulus
skyannulus_data       = apphot_annulus.to_mask (method='center').multiply (data)
skyannulus_mask       = skyannulus_data <= 0.0
skyannulus_maskeddata = numpy.ma.array (skyannulus_data, mask=skyannulus_mask)

# 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 (skyannulus_maskeddata, origin='lower', cmap=cmap)
fig.colorbar (im)

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

Run the script.

```
% chmod a+x advobs202202_s11_09.py
% ./advobs202202_s11_09.py -h
usage: advobs202202_s11_09.py [-h] [-f FWHM] [-a APERTURE] [-s1 SKYANNULUS1]
                              [-s2 SKYANNULUS2] [-w WIDTH] [-x XCENTRE]
                              [-y YCENTRE] [-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

extraction of pixel values within sky annulus

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  -f FWHM, --fwhm FWHM  FWHM of stellar PSF in pixel (default: 4.0)
  -a APERTURE, --aperture APERTURE
                        aperture radius in FWHM (default: 1.5)
  -s1 SKYANNULUS1, --skyannulus1 SKYANNULUS1
                        inner sky annulus radius in FWHM (default: 3.0)
  -s2 SKYANNULUS2, --skyannulus2 SKYANNULUS2
                        outer sky annulus radius in FWHM (default: 5.0)
  -w WIDTH, --width WIDTH
                        half-width of subframe to be plotted (default: 15)
  -x XCENTRE, --xcentre XCENTRE
                        x coordinate of target
  -y YCENTRE, --ycentre YCENTRE
                        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

% ./advobs202202_s11_09.py -f 5.03 -a 2.0 -s1 3.0 -s2 6.0 -w 35 \
? -x 500.01 -y 500.01 -m inferno -o skyannulus_0.png synstars_0.fits
% ls -lF skyannulus_0.png
-rw-r--r--  1 daisuke  taiwan  119125 Apr 14 15:47 skyannulus_0.png
```

Show the plot. (Fig. 11)

```
% feh -dF skyannulus_0.png
```

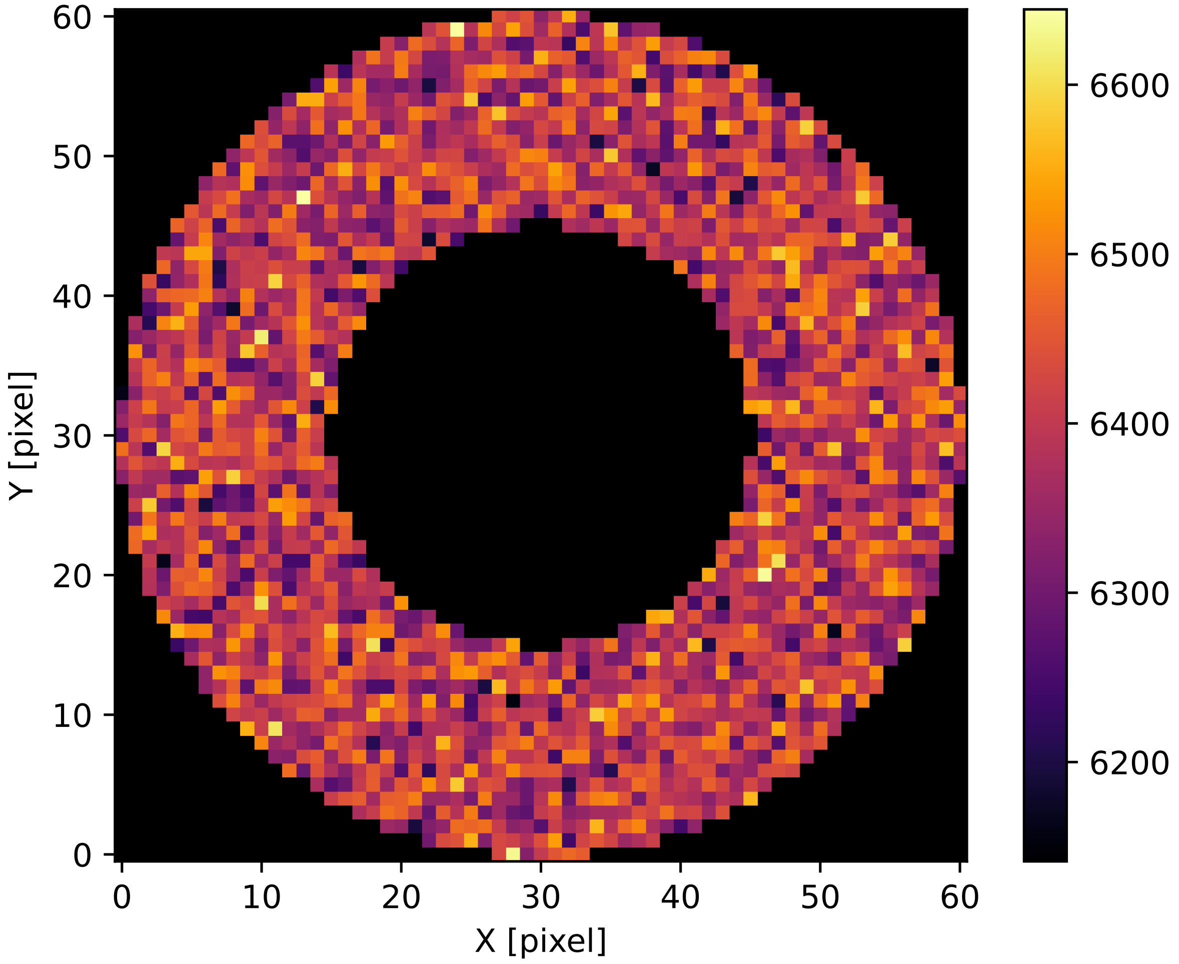


Figure 11: Extracted pixels within sky annulus.

4.7 Aperture photometry using smaller aperture radius

Use smaller radius for aperture, and do aperture photometry again.
Run the script using the aperture radius of 1.5 times of FWHM.

```
% ./advobs202202_s11_08.py -f 5.03 -a 1.5 -s1 3.0 -s2 6.0 -w 35 \
? -x 500.01 -y 500.01 synstars_0.fits
# now, reading FITS file 'synstars_0.fits'...
# finished reading FITS file 'synstars_0.fits'!
# now, generating an aperture...
aperture for star:
Aperture: CircularAperture
positions: [35.01, 35.01]
r: 7.545
sky annulus:
Aperture: CircularAnnulus
positions: [35.01, 35.01]
r_in: 15.09
r_out: 30.18
# finished generating an aperture!
# now, adding all the signal values within aperture...
 id      xcenter           ycenter          aperture_sum
           pix               pix
--- ----------------- ----------------- ------------------
  1 35.00999999999999 35.00999999999999 1444086.0666451398
aperture sum         = 1444086.066645 ADU
# finished adding all the signal values within aperture!
# now, estimating sky background value...
sky background       = 6400.339872 ADU/pix
sky background error = 79.350522 ADU/pix
# finished estimating sky background value!
# now, subtracting sky background...
# finished subtracting sky background!
#
# result of aperture photometry
#
# X, Y, APERTURE_RADIUS, OUTER_SKY_ANNULUS, INNER_SKY_ANNULUS,
# NPIX_APERTURE, FLUX, SKY_PER_PIX, SKY_ERR_PER_PIX,
# NET_FLUX, NET_FLUX_ERR
#
500.010000 500.010000 7.545000 15.090000 30.180000 178.841524 1444086.066645 640
0.339872 79.350522 299439.532808 1603.172688
```

Obtained value is 299,400 $\pm$ 1,600 ADU. Major contribution to the total error is the one from sky background estimate, and we get smaller total error for smaller aperture size.

5 For your further reading

1. Read chapter 5 of "Handbook of CCD Astronomy" and learn about aperture photometry.
   - Handbook of CCD Astronomy (2nd Edition)
     - Steve B. Howell
     - Cambridge University Press
     - `https://doi.org/10.1017/CBO9780511807909`
2. Read the paper "Basic Photometry Techniques" and learn about aperture photometry.
   - "Basic Photometry Techniques"

- Da Costa, G. S.
- Astronomical CCD observing and reduction techniques, edited by Steve B. Howell.
- 1992
- `http://adsabs.harvard.edu/full/1992ASPC...23...90D`

6 Assignment

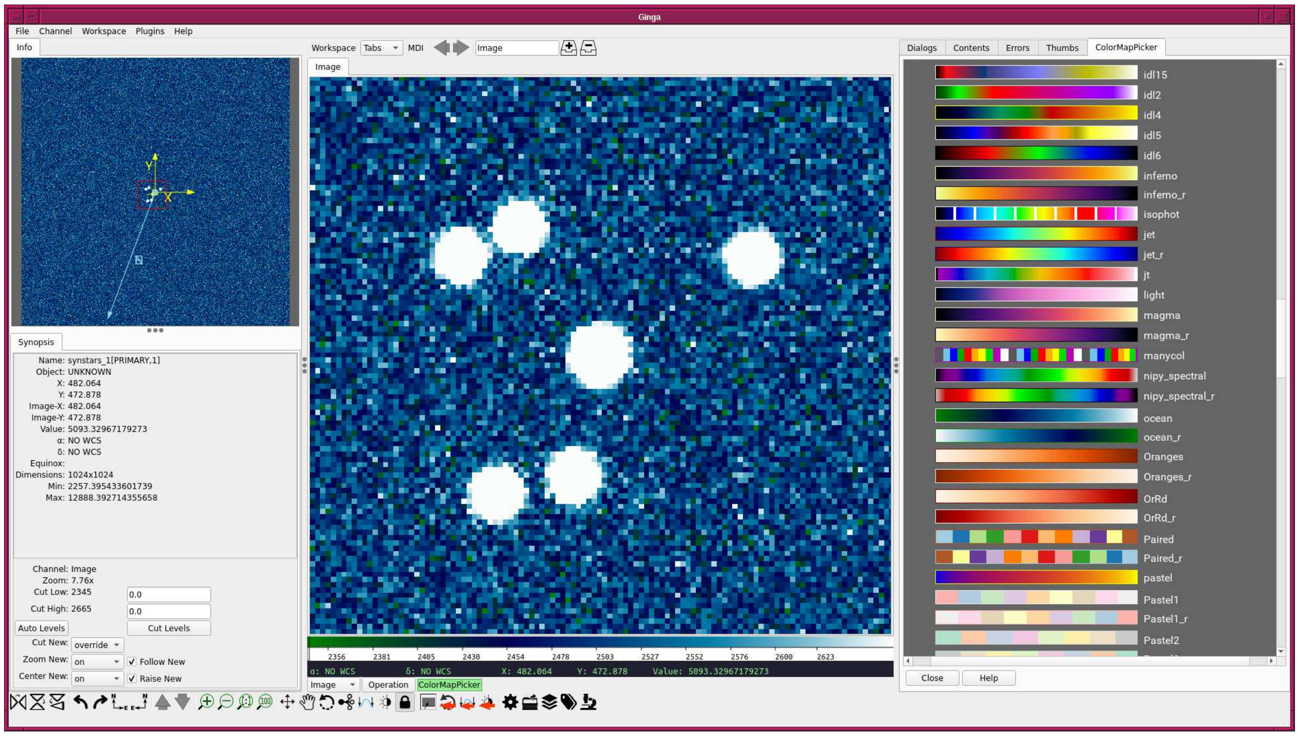
1. What is aperture photometry?
   - (a) Describe the method of aperture photometry. What is the basic idea of aperture photometry? Why do we need to set sky annulus in addition to aperture for star?
   - (b) What are pros and cons of aperture photometry?
   - (c) What are commonly used photometry techniques other than aperture photometry? Under what circumstances we should use photometry techniques other than aperture photometry?
2. Optical aperture size:
   - (a) What are pros and cons of larger aperture radius?
   - (b) What are pros and cons of smaller aperture radius?
   - (c) For brighter sources, should we use larger aperture raduis, or smaller aperture radius? Why? Describe your answer.
   - (d) Typically, 1.5 times of FWHM or 2.0 times of FWHM is a good choice for aperture radius. Describe why.
   - (e) For fainter sources, should we use larger aperture raduis, or smaller aperture radius? Why? Describe your answer.
3. For 24 artificial stars other than the one you have measured on the image `synstars_0.fits`, make your own Python script to carry out aperture photometry using `photutils`. Show the source code of your Python script. Execute the script, and show the results. Examine your results.
4. Making one more synthetic image and doing aperture photometry.
   - (a) Create a synthetic image using Python script `ao2021_s11_10.py`. The image should look like Fig. 12. You see fainter stars around the brighter star at (x, y) = (500, 500).
   - (b) Make your own Python script to carry out aperture photometry of a star centred on (x, y) = (500, 500). Use simple mean of pixel values within sky annulus to estimate sky background level. Show the source code of your Python script. Execute the script, and show the result of sky background level estimate and aperture photometry.
   - (c) Make your own Python script to carry out aperture photometry of a star centred on (x, y) = (500, 500). Use median of sigma clipped data of pixel values within sky annulus to estimate sky background level. Show the source code of your Python script. Execute the script, and show the result of sky background level estimate and aperture photometry.
   - (d) Does sigma-clipping algorithm give a better estimate of sky background level? If so, describe why.
5. Flux falling within the aperture:
   - (a) Assume 2-dimensional Gaussian function for the stellar PSF. When we set the aperture radius to be 1.0 times of FWHM, what is the fraction of stellar flux falling within the aperture? Make a Python script to calculate it. Show the source code of your Python script. Execute the script, and show the result.
   - (b) Assume 2-dimensional Gaussian function for the stellar PSF. When we set the aperture radius to be 1.5 times of FWHM, what is the fraction of stellar flux falling within the aperture? Make a Python script to calculate it. Show the source code of your Python script. Execute the script, and show the result.
   - (c) Assume 2-dimensional Gaussian function for the stellar PSF. When we set the aperture radius to be 2.0 times of FWHM, what is the fraction of stellar flux falling within the aperture? Make a Python script to calculate it. Show the source code of your Python script. Execute the script, and show the result.
   - (d) Assume 2-dimensional Gaussian function for the stellar PSF. When we set the aperture radius to be 2.5 times of FWHM, what is the fraction of stellar flux falling within the aperture? Make a Python script to calculate it. Show the source code of your Python script. Execute the script, and show the result.

Python Code 10: advobs202202_s11_10.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# 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=2500.0, \
                     help='background level (default: 2500)')
parser.add_argument ('-s', '--sigma', type=float, default=50.0, \
                     help='noise level (default: 50)')
parser.add_argument ('-n', '--nstar', type=int, default=10, \
                     help='number of stars to generate (default: 10)')
parser.add_argument ('-f1', '--flux1', type=float, default=100000.0, \
                     help='total flux of bright star (default: 100000)')
parser.add_argument ('-f2', '--flux2', type=float, default=30000.0, \
                     help='total flux of faint star (default: 30000)')
parser.add_argument ('-p', '--psf', type=float, default=5.0, \
                     help='FWHM of stellar radial profile (default: 5)')
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_total_1         = args.flux1
flux_total_2         = args.flux2
psf_fwhm             = args.psf
file_output          = args.output

# checking output file name
if (file_output == ''):
    print ("You need to specify output file name.")
    sys.exit ()
if not (file_output[-5:] == '.fits'):
    print ("Output file must be a FITS file.")
    sys.exit ()
```

```
# image size
image_size_x = 1024
image_size_y = 1024
image_size   = (image_size_x, image_size_y)

# location of bright star
x_centre = 500.0
y_centre = 500.0

# region to make stars
half_width = 30.0
x_min = x_centre - half_width
x_max = x_centre + half_width
y_min = y_centre - half_width
y_max = y_centre + half_width

list_x    = numpy.array ([x_centre])
list_y    = numpy.array ([y_centre])
list_flux = numpy.array ([flux_total_1])
list_psf  = numpy.array ([psf_fwhm])

while ( len (list_x) < nstar + 1 ):
    x = numpy.random.uniform (x_min, x_max)
    y = numpy.random.uniform (y_min, y_max)
    dist = numpy.sqrt ( (x - x_centre)**2 + (y - y_centre)**2 )
    if (dist < psf_fwhm * 3.0):
        continue
    list_x    = numpy.append (list_x, x)
    list_y    = numpy.append (list_y, y)
    list_flux = numpy.append (list_flux, flux_total_2)
    list_psf  = numpy.append (list_psf, psf_fwhm)

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

# 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']   = list_x
source_table['y_0']   = list_y
source_table['flux']  = list_flux
source_table['sigma'] = list_psf / fwhm_sigma

# printing source table
print ("source_table:")
print (source_table)

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

# making synthetic image by adding background and stars
image = image_background + image_star
```

```
# writing a FITS file
print ("Now, writing data into FITS file \"%s\"..." % file_output)
astropy.io.fits.writeto (file_output, image)
print ("Finished writing data!")
```



```
Name: synstars_1[PRIMARY,1]
Object: UNKNOWN
X: 482.064
Y: 472.878
Image-X: 482.064
Image-Y: 472.878
Value: 5093.32967179273
α: NO WCS
δ: NO WCS
Equinox:
Dimensions: 1024x1024
Min: 2257.395433601739
Max: 12888.392714355658

Channel: Image
Zoom: 7.76x
Cut Low: 2345      0.0
Cut High: 2665     0.0
Auto Levels        Cut Levels
Cut New: override
Zoom New: on       ✓ Follow New
Center New: on     ✓ Raise New

α: NO WCS   δ: NO WCS   X: 482.064   Y: 472.878   Value: 5093.32967179273
```

Figure 12: The synthetic image created by the script ao2021_s11_10.py.