Advanced Astronomical Observations 2022
Session 16: Source Extraction

Kinoshita Daisuke

26 May 2021
publicly accessible version

About this file...

- Important information about this file
  - The author of this file is Kinoshita Daisuke.
  - The original version of this file was used for the course "Advanced Astronomical Observations" (course ID: AS6005) offered at Institute of Astronomy, National Central University from February 2022 to June 2022.
  - The file is provided in the hope that it will be useful, but there is no guarantee for the correctness. Use this file at your own risk.
  - If you are willing to use this file for your study, please feel free to use. I'll be very happy to receive feedback from you.
  - If you are willing to use this file for your teaching, please contact to Kinoshita Daisuke. When you use this file partly or entirely, please mention clearly that the author of the original version is Kinoshita Daisuke. Please help me to improve the contents of this file by sending your feedback.
  - Contact address: `https://www.instagram.com/daisuke23888/`

For this session, we try source extraction.

1 Generating synthetic data

Make a Python script to generate a synthetic image with artificial stars and galaxies.

Python Code 1: advobs202202_s16_01.py

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

#
# Time-stamp: <2022/05/26 21:19:57 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing numpy module
import numpy
import numpy.random
```

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

# importing photutils module
import photutils.datasets

# constructing parser object
desc   = 'generating a synthetic image with artificial stars and galaxies'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-x', '--size-x', type=int, default=2048, \
                     help='image size in X-axis (default: 2048)')
parser.add_argument ('-y', '--size-y', type=int, default=2048, \
                     help='image size in Y-axis (default: 2048)')
parser.add_argument ('-n', '--nstars', type=int, default=100, \
                     help='number of stars to generate (default: 100)')
parser.add_argument ('-g', '--ngalaxies', type=int, default=10, \
                     help='number of galaxies to generate (default: 10)')
parser.add_argument ('-f', '--flux-min', type=float, default=1000.0, \
                     help='minimum flux of stars (default: 1000)')
parser.add_argument ('-p', '--fwhm-psf', type=float, default=3.5, \
                     help='FWHM of PSF in pixel (default: 3.5)')
parser.add_argument ('-d', '--fwhm-stddev', type=float, default=0.1, \
                     help='stddev of FWHM distribution in pixel (default: 0.1)')
parser.add_argument ('-q', '--fwhm-psf-gal', type=float, default=8.0, \
                     help='FWHM of galaxy PSF in pixel (default: 8)')
parser.add_argument ('-r', '--fwhm-psf-gal-stddev', type=float, default=4.0, \
                     help='stddev of FWHM of galaxy PSF in pixel (default: 4)')
parser.add_argument ('-s', '--sky', type=float, default=1000.0, \
                     help='sky background level in ADU (default: 1000)')
parser.add_argument ('-e', '--sky-stddev', type=float, default=30.0, \
                     help='stddev of sky background in ADU (default: 30)')
parser.add_argument ('-o', '--output-file', default='', \
                     help='output file name')
parser.add_argument ('-l', '--log-file', default='', \
                     help='log file name')

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

# image size
image_size_x = args.size_x
image_size_y = args.size_y
image_shape  = (image_size_x, image_size_y)

# number of stars and galaxies to generate
nstars = args.nstars
ngals  = args.ngalaxies

# flux of faintest stars
flux_min = args.flux_min

# FWHM of PSF
fwhm_x            = args.fwhm_psf
fwhm_y            = args.fwhm_psf
fwhm_stddev_x     = args.fwhm_stddev
fwhm_stddev_y     = args.fwhm_stddev
```

```
fwhm_gal_x        = args.fwhm_psf_gal
fwhm_gal_y        = args.fwhm_psf_gal
fwhm_gal_stddev_x = args.fwhm_psf_gal_stddev
fwhm_gal_stddev_y = args.fwhm_psf_gal_stddev

# sky background level and stddev
sky_mean   = args.sky
sky_stddev = args.sky_stddev

# output file name and log file name
file_output = args.output_file
file_log    = args.log_file

# making pathlib objects
path_output = pathlib.Path (file_output)
path_log    = pathlib.Path (file_log)

# check of output file name
if not (path_output.suffix == '.fits'):
    # printing message
    print ("ERROR: Output file must be a FITS file.")
    # exit
    sys.exit ()

# check of log file name
if (file_log == ''):
    # printing message
    print ("ERROR: You need to specify log file name.")
    # exit
    sys.exit ()

# existence check for output file
if (path_output.exists ()):
    # printing message
    print ("ERROR: output file '%s' exists." % (file_output) )
    # exit
    sys.exit ()

# existence check for log file
if (path_log.exists ()):
    # printing message
    print ("ERROR: log file '%s' exists." % (file_log) )
    # exit
    sys.exit ()

# generating a new astropy table
table_stars = astropy.table.Table ()
table_gals  = astropy.table.Table ()

# random number generator
rng = numpy.random.default_rng ()

# generating random numbers for stars
position_x = rng.uniform (0, image_size_x, nstars)
position_y = rng.uniform (0, image_size_y, nstars)
theta_deg  = rng.uniform (0, 360, nstars)
psf_x      = rng.normal (loc=fwhm_x, scale=fwhm_stddev_x, size=nstars)
psf_y      = rng.normal (loc=fwhm_y, scale=fwhm_stddev_y, size=nstars)
powerlaw   = rng.power (1.5, size=nstars)
```

```
flux       = flux_min / powerlaw

# generating random numbers for galaxies
centre_gal_x   = rng.uniform (image_size_x * 0.3, image_size_x * 0.7)
centre_gal_y   = rng.uniform (image_size_y * 0.3, image_size_y * 0.7)
position_gal_x = rng.normal (loc=centre_gal_x, scale=300, size=ngals)
position_gal_y = rng.normal (loc=centre_gal_y, scale=300, size=ngals)
theta_gal_deg  = rng.uniform (0, 360, ngals)
psf_gal_x      = rng.normal (loc=fwhm_gal_x, scale=fwhm_gal_stddev_x, \
                             size=ngals)
psf_gal_y      = rng.normal (loc=fwhm_gal_y, scale=fwhm_gal_stddev_y, \
                             size=ngals)
powerlaw_gal   = rng.power (2.0, size=ngals)
flux_gal       = flux_min * 3 / powerlaw_gal

# conversion from degree to radian
theta_rad     = numpy.radians (theta_deg)
theta_gal_rad = numpy.radians (theta_gal_deg)

# adding data to the table of stars
table_stars['amplitude'] = flux
table_stars['x_mean']    = position_x
table_stars['y_mean']    = position_y
table_stars['x_stddev']  = psf_x
table_stars['y_stddev']  = psf_y
table_stars['theta']     = theta_rad

# adding data to the table of galaxies
table_gals['amplitude'] = flux_gal
table_gals['x_mean']    = position_gal_x
table_gals['y_mean']    = position_gal_y
table_gals['x_stddev']  = psf_gal_x
table_gals['y_stddev']  = psf_gal_y
table_gals['theta']     = theta_gal_rad

# writing positions of stars and galaxies to log file
with open (file_log, 'w') as fh_log:
    # information of stars
    fh_log.write ("#\n")
    fh_log.write ("# input parameters for producing synthetic image\n")
    fh_log.write ("#\n")
    fh_log.write ("#   image_size_x      = %d\n" % image_size_x)
    fh_log.write ("#   image_size_y      = %d\n" % image_size_y)
    fh_log.write ("#   nstars            = %d\n" % nstars)
    fh_log.write ("#   ngals             = %d\n" % ngals)
    fh_log.write ("#   flux_min          = %f\n" % flux_min)
    fh_log.write ("#   fwhm_x            = %f\n" % fwhm_x)
    fh_log.write ("#   fwhm_y            = %f\n" % fwhm_y)
    fh_log.write ("#   fwhm_stddev_x     = %f\n" % fwhm_stddev_x)
    fh_log.write ("#   fwhm_stddev_y     = %f\n" % fwhm_stddev_y)
    fh_log.write ("#   fwhm_gal_x        = %f\n" % fwhm_gal_x)
    fh_log.write ("#   fwhm_gal_y        = %f\n" % fwhm_gal_y)
    fh_log.write ("#   fwhm_gal_stddev_x = %f\n" % fwhm_gal_stddev_x)
    fh_log.write ("#   fwhm_gal_stddev_y = %f\n" % fwhm_gal_stddev_y)
    fh_log.write ("#   sky_mean          = %f\n" % sky_mean)
    fh_log.write ("#   sky_stddev        = %f\n" % sky_stddev)
    fh_log.write ("#   file_output       = %s\n" % file_output)
    fh_log.write ("#   file_log          = %s\n" % file_log)
    fh_log.write ("#\n")
```

```
    fh_log.write ("# information of stars\n")
    fh_log.write ("#\n")
    astropy.io.ascii.write (table_stars, fh_log, format='commented_header')
    # information of galaxies
    fh_log.write ("#\n")
    fh_log.write ("# information of galaxies\n")
    fh_log.write ("#\n")
    astropy.io.ascii.write (table_gals, fh_log, format='commented_header')

# generating stars
image_stars = photutils.datasets.make_gaussian_sources_image (image_shape, \
                                                               table_stars)

# generating galaxies
image_gals = photutils.datasets.make_gaussian_sources_image (image_shape, \
                                                             table_gals)

# generating sky background
image_sky = photutils.datasets.make_noise_image (image_shape, \
                                                 distribution='gaussian', \
                                                 mean=sky_mean, \
                                                 stddev=sky_stddev)
# generating synthetic image
image = image_stars + image_gals + image_sky

# making WCS
wcs = photutils.datasets.make_wcs (image_shape)

# making Image HDU
hdu = photutils.datasets.make_imagehdu (image, wcs=wcs)

# writing a FITS file
hdu.writeto (file_output)
```

Execute the script, and generate a synthetic image.

```
% chmod a+x advobs202202_s16_01.py
% ./advobs202202_s16_01.py -h
usage: advobs202202_s16_01.py [-h] [-x SIZE_X] [-y SIZE_Y] [-n NSTARS]
                              [-g NGALAXIES] [-f FLUX_MIN] [-p FWHM_PSF]
                              [-d FWHM_STDDEV] [-q FWHM_PSF_GAL]
                              [-r FWHM_PSF_GAL_STDDEV] [-s SKY]
                              [-e SKY_STDDEV] [-o OUTPUT_FILE] [-l LOG_FILE]

generating a synthetic image with artificial stars and galaxies

optional arguments:
  -h, --help            show this help message and exit
  -x SIZE_X, --size-x SIZE_X
                        image size in X-axis (default: 2048)
  -y SIZE_Y, --size-y SIZE_Y
                        image size in Y-axis (default: 2048)
  -n NSTARS, --nstars NSTARS
                        number of stars to generate (default: 100)
  -g NGALAXIES, --ngalaxies NGALAXIES
                        number of galaxies to generate (default: 10)
  -f FLUX_MIN, --flux-min FLUX_MIN
                        minimum flux of stars (default: 1000)
```

```
  -p FWHM_PSF, --fwhm-psf FWHM_PSF
                        FWHM of PSF in pixel (default: 3.5)
  -d FWHM_STDDEV, --fwhm-stddev FWHM_STDDEV
                        stddev of FWHM distribution in pixel (default: 0.1)
  -q FWHM_PSF_GAL, --fwhm-psf-gal FWHM_PSF_GAL
                        FWHM of galaxy PSF in pixel (default: 8)
  -r FWHM_PSF_GAL_STDDEV, --fwhm-psf-gal-stddev FWHM_PSF_GAL_STDDEV
                        stddev of FWHM of galaxy PSF in pixel (default: 4)
  -s SKY, --sky SKY     sky background level in ADU (default: 1000)
  -e SKY_STDDEV, --sky-stddev SKY_STDDEV
                        stddev of sky background in ADU (default: 30)
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        output file name
  -l LOG_FILE, --log-file LOG_FILE
                        log file name

% ./advobs202202_s16_01.py -n 200 -g 30 -l synthetic_0.log -o synthetic_0.fits
% ls -lF synthetic_0.*
-rw-r--r--  1 daisuke  taiwan  33560640 May 26 21:21 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan     26313 May 26 21:20 synthetic_0.log
% head -25 synthetic_0.log
#
# input parameters for producing synthetic image
#
#   image_size_x      = 2048
#   image_size_y      = 2048
#   nstars            = 200
#   ngals             = 30
#   flux_min          = 1000.000000
#   fwhm_x            = 3.500000
#   fwhm_y            = 3.500000
#   fwhm_stddev_x     = 0.100000
#   fwhm_stddev_y     = 0.100000
#   fwhm_gal_x        = 8.000000
#   fwhm_gal_y        = 8.000000
#   fwhm_gal_stddev_x = 4.000000
#   fwhm_gal_stddev_y = 4.000000
#   sky_mean          = 1000.000000
#   sky_stddev        = 30.000000
#   file_output       = synthetic_0.fits
#   file_log          = synthetic_0.log
#
# information of stars
#
# amplitude x_mean y_mean x_stddev y_stddev theta
1883.5484688100564 842.6175788096168 1136.0531362803365 3.4586606356120293 3.510
414047690422 1.735543357533016
```

Use Ginga to check the image. (Fig. 1)

```
% ginga ginga synthetic_0.fits &
```

2 Background estimation

Make a Python script to estimate background level of an image.

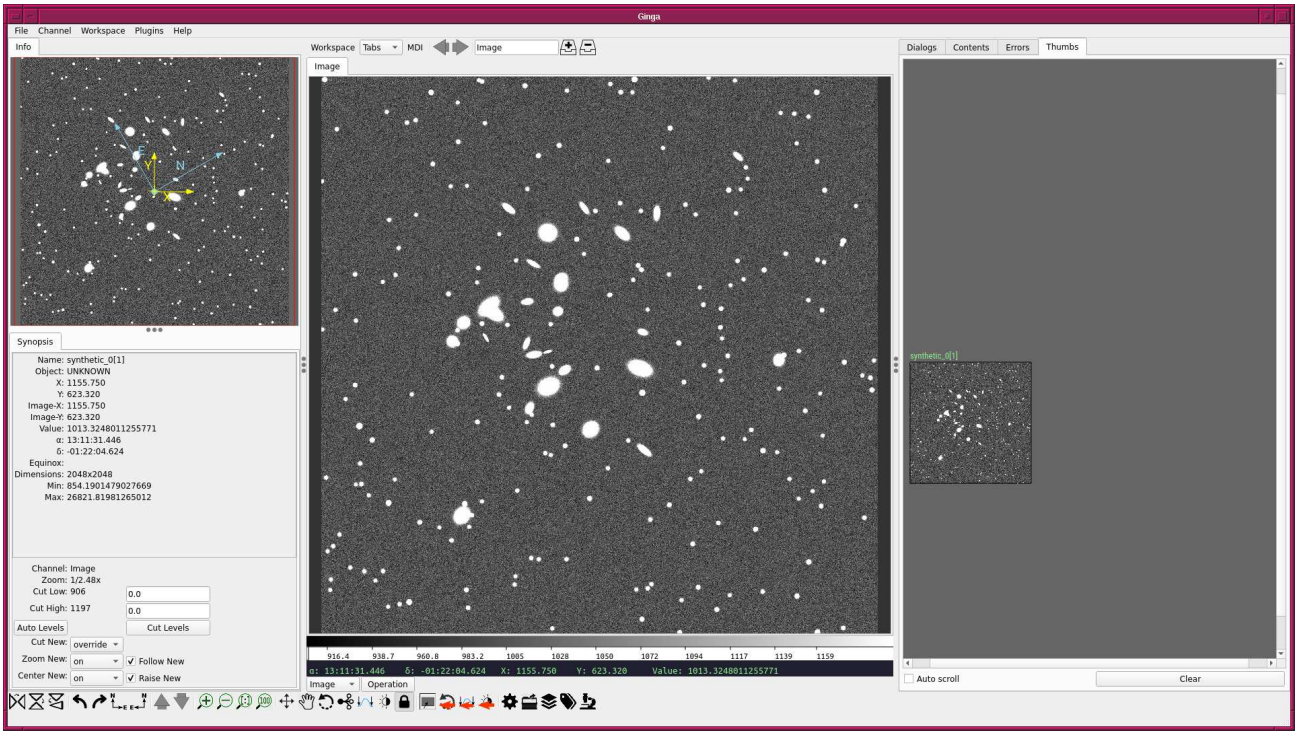

Figure 1: A synthetic image of artificial stars and galaxies.

Python Code 2: advobs202202_s16_02.py

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

#
# Time-stamp: <2022/05/26 21:32:13 (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.ma

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

# importing photutils module
import photutils.segmentation

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

```
# constructing parser object
desc   = 'background estimation using source detection and masking'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input-file', default='', \
                     help='input file name')
parser.add_argument ('-t', '--threshold', type=float, default=2.0, \
                     help='detection threshold in sigma (default: 2)')
parser.add_argument ('-n', '--npixels', type=int, default=5, \
                     help='minimum number of pixels for detection (default: 5)')
parser.add_argument ('-s', '--dilate-size', type=int, default=21, \
                     help='dilate size (default: 21)')
parser.add_argument ('-m', '--maxiters', type=int, default=30, \
                     help='maximum number of iterations (default: 30)')
parser.add_argument ('-r', '--sigma-clipping', type=float, default=4.0, \
                     help='sigma-clipping threshold in sigma (default: 4)')

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

# file names
file_input  = args.input_file

# input parameters
threshold   = args.threshold
npixels     = args.npixels
dilate_size = args.dilate_size
maxiters    = args.maxiters
rejection   = args.sigma_clipping

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

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

# existence check of input file
if not (path_input.exists ()):
    # printing message
    print ("ERROR: Input file '%s' does not exist." % (file_input) )
    # exit
    sys.exit ()

# opening FITS file
with astropy.io.fits.open (file_input) as hdu:
    # reading header and image
    header = hdu[0].header
    image  = hdu[0].data
    # if no image in PrimaryHDU, then read next HDU
    if (header['NAXIS'] == 0):
        header = hdu[1].header
        image  = hdu[1].data
```

```
# making source mask
source_mask \
    = photutils.segmentation.make_source_mask (image, threshold,
                                               npixels=npixels,
                                               sigclip_iters=maxiters,
                                               dilate_size=dilate_size)

# making masked array
image_masked = numpy.ma.array (image, mask=source_mask)

# sigma-clipping
skybg_mean, skybg_median, skybg_stddev \
    = astropy.stats.sigma_clipped_stats (image, sigma=rejection)

# mode calculation using empirical formula
skybg_mode = 3.0 * skybg_median - 2.0 * skybg_mean

# printing results
print ("#")
print ("# background estimation using source detection and masking")
print ("#")
print ("# date/time = %s" % now)
print ("#")
print ("# input parameters")
print ("#")
print ("#   input file                          = %s" % file_input)
print ("#   detection threshold                 = %f sigma" % threshold)
print ("#   min number of pixels for detection = %d pixel" % npixels)
print ("#   dilate size                         = %d pixel" % dilate_size)
print ("#   max number of iterations            = %d" % maxiters)
print ("#   sigma-clipping threshold            = %f sigma" % rejection)
print ("#")
print ("# results")
print ("#")
print ("#   mean, median, mode, stddev")
print ("#")
print ("%f %f %f %f" % (skybg_mean, skybg_median, skybg_mode, skybg_stddev) )
```

Run the script, and check the estimated sky background level.

```
% chmod a+x advobs202202_s16_02.py
% ./advobs202202_s16_02.py -h
usage: advobs202202_s16_02.py [-h] [-i INPUT_FILE] [-t THRESHOLD] [-n NPIXELS]
                              [-s DILATE_SIZE] [-m MAXITERS]
                              [-r SIGMA_CLIPPING]

background estimation using source detection and masking

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT_FILE, --input-file INPUT_FILE
                        input file name
  -t THRESHOLD, --threshold THRESHOLD
                        detection threshold in sigma (default: 2)
  -n NPIXELS, --npixels NPIXELS
                        minimum number of pixels for detection (default: 5)
  -s DILATE_SIZE, --dilate-size DILATE_SIZE
                        dilate size (default: 21)
```

```
  -m MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 30)
  -r SIGMA_CLIPPING, --sigma-clipping SIGMA_CLIPPING
                        sigma-clipping threshold in sigma (default: 4)

% ./advobs202202_s16_02.py -i synthetic_0.fits > synthetic_0.sky
% ls -lF synthetic_0.*
-rw-r--r--  1 daisuke  taiwan  33560640 May 26 21:21 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan     26313 May 26 21:20 synthetic_0.log
-rw-r--r--  1 daisuke  taiwan       532 May 26 21:34 synthetic_0.sky
% cat synthetic_0.sky
#
# background estimation using source detection and masking
#
# date/time = 2022-05-26 21:34:24.339569
#
# input parameters
#
#   input file                          = synthetic_0.fits
#   detection threshold                 = 2.000000 sigma
#   min number of pixels for detection = 5 pixel
#   dilate size                         = 21 pixel
#   max number of iterations            = 30
#   sigma-clipping threshold            = 4.000000 sigma
#
# results
#
#   mean, median, mode, stddev
#
1000.810064 1000.535514 999.986415 30.673467
```

Estimated sky background level is 999.99 ADU, and it is very close to expected value of 1000 ADU.

3 Source detection using image segmentation

Make a Python script to carry out source detection using image segmentation.

Python Code 3: advobs202202_s16_03.py

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

#
# Time-stamp: <2022/05/26 21:42:48 (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.ma

# importing astropy module
import astropy.convolution
import astropy.io.fits
import astropy.stats
import astropy.visualization
import astropy.visualization.mpl_normalize

# importing photutils module
import photutils.segmentation

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

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

# constructing parser object
desc   = 'source extraction using image segmentation'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input-file', default='', \
                     help='input file name')
parser.add_argument ('-o', '--output-file', default='', \
                     help='output file name')
parser.add_argument ('-t', '--threshold', type=float, default=3.0, \
                     help='detection threshold in sigma (default: 3)')
parser.add_argument ('-u', '--threshold-for-sky', type=float, default=2.0, \
                     help='detection threshold for sky estimate (default: 2)')
parser.add_argument ('-n', '--npixels', type=int, default=5, \
                     help='minimum number of pixels for detection (default: 5)')
parser.add_argument ('-s', '--dilate-size', type=int, default=21, \
                     help='dilate size (default: 21)')
parser.add_argument ('-m', '--maxiters', type=int, default=30, \
                     help='maximum number of iterations (default: 30)')
parser.add_argument ('-r', '--sigma-clipping', type=float, default=4.0, \
                     help='sigma-clipping threshold in sigma (default: 4)')
parser.add_argument ('-k', '--gaussian-fwhm', type=float, default=3.0, \
                     help='Gaussian FWHM in pixel for convolution (default: 3)')
parser.add_argument ('-a', '--kernel-size', type=int, default=3, \
                     help='Gaussian kernel array size in pixel (default: 3)')

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

# file names
file_input  = args.input_file
file_output = args.output_file

# input parameters
threshold         = args.threshold
threshold_for_sky = args.threshold
npixels           = args.npixels
dilate_size       = args.dilate_size
maxiters          = args.maxiters
rejection         = args.sigma_clipping
```

```
gaussian_fwhm     = args.gaussian_fwhm
kernel_array_size = args.kernel_size

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

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

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

# existence check of input file
if not (path_input.exists ()):
    # printing message
    print ("ERROR: Input file '%s' does not exist." % (file_input) )
    # exit
    sys.exit ()

# existence check of output file
if (path_output.exists ()):
    # printing message
    print ("ERROR: Output file '%s' exists." % (file_output) )
    # exit
    sys.exit ()

# opening FITS file
with astropy.io.fits.open (file_input) as hdu:
    # reading header and image
    header = hdu[0].header
    image  = hdu[0].data
    # if no image in PrimaryHDU, then read next HDU
    if (header['NAXIS'] == 0):
        header = hdu[1].header
        image  = hdu[1].data

# making source mask
source_mask \
    = photutils.segmentation.make_source_mask (image, threshold_for_sky,
                                               npixels=npixels,
                                               sigclip_iters=maxiters,
                                               dilate_size=dilate_size)

# making masked array
image_masked = numpy.ma.array (image, mask=source_mask)

# sigma-clipping
skybg_mean, skybg_median, skybg_stddev \
    = astropy.stats.sigma_clipped_stats (image, sigma=rejection)
```

```
# mode calculation using empirical formula
skybg_mode = 3.0 * skybg_median - 2.0 * skybg_mean

# detection threshold in ADU
threshold_adu = skybg_mode + threshold * skybg_stddev

# 2D Gaussian kernel for convolution
gaussian_sigma = gaussian_fwhm * astropy.stats.gaussian_fwhm_to_sigma
kernel = astropy.convolution.Gaussian2DKernel (gaussian_sigma, \
                                               x_size=kernel_array_size, \
                                               y_size=kernel_array_size)
kernel.normalize ()

# source detection
image_segm = photutils.segmentation.detect_sources (image, threshold_adu, \
                                                    npixels=npixels, \
                                                    kernel=kernel)

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax1 = fig.add_subplot (121)
ax2 = fig.add_subplot (122)

# normalisation
norm \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image) )

# plotting original image
im1 = ax1.imshow (image, origin='upper', cmap='viridis', norm=norm)
ax1.set_title ('Original Image')

# plotting segmentation image
im2 = ax2.imshow (image_segm, origin='upper', \
                  cmap=image_segm.make_cmap (), interpolation='nearest')
ax2.set_title ('Segmentation Image')

# writing to a file
fig.savefig (file_output, dpi=225)
```

Execute the script, and make a segmentation image.

```
% chmod a+x advobs202202_s16_03.py
% ./advobs202202_s16_03.py -h
usage: advobs202202_s16_03.py [-h] [-i INPUT_FILE] [-o OUTPUT_FILE]
                              [-t THRESHOLD] [-u THRESHOLD_FOR_SKY]
                              [-n NPIXELS] [-s DILATE_SIZE] [-m MAXITERS]
                              [-r SIGMA_CLIPPING] [-k GAUSSIAN_FWHM]
                              [-a KERNEL_SIZE]

source extraction using image segmentation

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT_FILE, --input-file INPUT_FILE
                        input file name
```

```
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        output file name
  -t THRESHOLD, --threshold THRESHOLD
                        detection threshold in sigma (default: 3)
  -u THRESHOLD_FOR_SKY, --threshold-for-sky THRESHOLD_FOR_SKY
                        detection threshold for sky estimate (default: 2)
  -n NPIXELS, --npixels NPIXELS
                        minimum number of pixels for detection (default: 5)
  -s DILATE_SIZE, --dilate-size DILATE_SIZE
                        dilate size (default: 21)
  -m MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 30)
  -r SIGMA_CLIPPING, --sigma-clipping SIGMA_CLIPPING
                        sigma-clipping threshold in sigma (default: 4)
  -k GAUSSIAN_FWHM, --gaussian-fwhm GAUSSIAN_FWHM
                        Gaussian FWHM in pixel for convolution (default: 3)
  -a KERNEL_SIZE, --kernel-size KERNEL_SIZE
                        Gaussian kernel array size in pixel (default: 3)

% ./advobs202202_s16_03.py -i synthetic_0.fits -o segm_0.pdf
% ls -lF segm_0.pdf
-rw-r--r--  1 daisuke  taiwan  222045 May 26 21:43 segm_0.pdf
```

Show the segmentation image. Sources are successfully detected. (Fig. 2)

```
% okular segm_0.pdf
```

4 Deblending

Make a Python script to carry out source deblending to separate overlapping sources.

Python Code 4: advobs202202_s16_04.py

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

#
# Time-stamp: <2022/05/26 21:49:10 (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.ma

# importing astropy module
import astropy.convolution
```

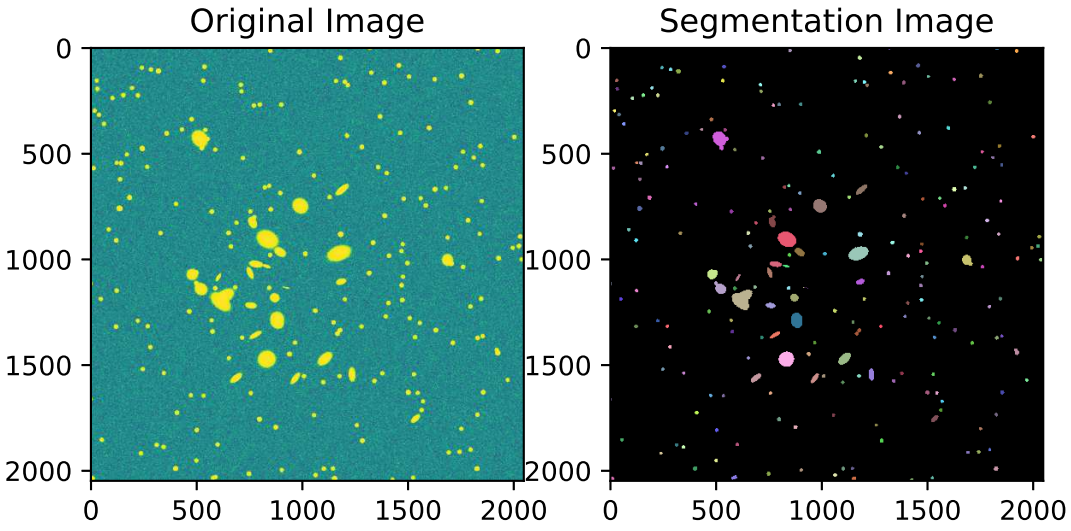

Figure 2: Segmantation image produced by photutils package.

```
import astropy.io.fits
import astropy.stats
import astropy.visualization
import astropy.visualization.mpl_normalize

# importing photutils module
import photutils.segmentation

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

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

# constructing parser object
desc   = 'source extraction using image segmentation and deblending'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input-file', default='', \
                     help='input file name')
parser.add_argument ('-o', '--output-file', default='', \
                     help='output file name')
parser.add_argument ('-t', '--threshold', type=float, default=3.0, \
                     help='detection threshold in sigma (default: 3)')
parser.add_argument ('-u', '--threshold-for-sky', type=float, default=2.0, \
                     help='detection threshold for sky estimate (default: 2)')
parser.add_argument ('-n', '--npixels', type=int, default=5, \
                     help='minimum number of pixels for detection (default: 5)')
parser.add_argument ('-s', '--dilate-size', type=int, default=21, \
                     help='dilate size (default: 21)')
parser.add_argument ('-m', '--maxiters', type=int, default=30, \
                     help='maximum number of iterations (default: 30)')
parser.add_argument ('-r', '--sigma-clipping', type=float, default=4.0, \
                     help='sigma-clipping threshold in sigma (default: 4)')
parser.add_argument ('-k', '--gaussian-fwhm', type=float, default=3.0, \
                     help='Gaussian FWHM in pixel for convolution (default: 3)')
parser.add_argument ('-a', '--kernel-size', type=int, default=3, \
                     help='Gaussian kernel array size in pixel (default: 3)')

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

# file names
file_input  = args.input_file
file_output = args.output_file

# input parameters
threshold         = args.threshold
threshold_for_sky = args.threshold
npixels           = args.npixels
dilate_size       = args.dilate_size
maxiters          = args.maxiters
rejection         = args.sigma_clipping
gaussian_fwhm     = args.gaussian_fwhm
kernel_array_size = args.kernel_size

# making pathlib objects
```

```
path_input  = pathlib.Path (file_input)
path_output = pathlib.Path (file_output)

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

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

# existence check of input file
if not (path_input.exists ()):
    # printing message
    print ("ERROR: Input file '%s' does not exist." % (file_input) )
    # exit
    sys.exit ()

# existence check of output file
if (path_output.exists ()):
    # printing message
    print ("ERROR: Output file '%s' exists." % (file_output) )
    # exit
    sys.exit ()

# opening FITS file
with astropy.io.fits.open (file_input) as hdu:
    # reading header and image
    header = hdu[0].header
    image  = hdu[0].data
    # if no image in PrimaryHDU, then read next HDU
    if (header['NAXIS'] == 0):
        header = hdu[1].header
        image  = hdu[1].data

# making source mask
source_mask \
    = photutils.segmentation.make_source_mask (image, threshold_for_sky,
                                               npixels=npixels,
                                               sigclip_iters=maxiters,
                                               dilate_size=dilate_size)

# making masked array
image_masked = numpy.ma.array (image, mask=source_mask)

# sigma-clipping
skybg_mean, skybg_median, skybg_stddev \
    = astropy.stats.sigma_clipped_stats (image, sigma=rejection)

# mode calculation using empirical formula
skybg_mode = 3.0 * skybg_median - 2.0 * skybg_mean
```

```
# detection threshold in ADU
threshold_adu = skybg_mode + threshold * skybg_stddev

# 2D Gaussian kernel for convolution
gaussian_sigma = gaussian_fwhm * astropy.stats.gaussian_fwhm_to_sigma
kernel = astropy.convolution.Gaussian2DKernel (gaussian_sigma, \
                                               x_size=kernel_array_size, \
                                               y_size=kernel_array_size)
kernel.normalize ()

# source detection
image_segm = photutils.segmentation.detect_sources (image, threshold_adu, \
                                                    npixels=npixels, \
                                                    kernel=kernel)

# deblending
image_deblend = photutils.segmentation.deblend_sources (image, image_segm, \
                                                        npixels=npixels, \
                                                        kernel=kernel, \
                                                        nlevels=32, \
                                                        contrast=0.001)

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax1 = fig.add_subplot (121)
ax2 = fig.add_subplot (122)

# plotting segmentation image
im1 = ax1.imshow (image_segm, origin='upper', \
                  cmap=image_segm.make_cmap (), interpolation='nearest')
ax1.set_title ('Segmentation Image')

# plotting deblended image
im2 = ax2.imshow (image_deblend, origin='upper', \
                  cmap=image_segm.make_cmap (), interpolation='nearest')
ax2.set_title ('Deblended Image')

# writing to a file
fig.savefig (file_output, dpi=225)
```

Run the script, and try deblending.

```
% chmod a+x advobs202202_s16_04.py
% ./advobs202202_s16_04.py -h
usage: advobs202202_s16_04.py [-h] [-i INPUT_FILE] [-o OUTPUT_FILE]
                              [-t THRESHOLD] [-u THRESHOLD_FOR_SKY]
                              [-n NPIXELS] [-s DILATE_SIZE] [-m MAXITERS]
                              [-r SIGMA_CLIPPING] [-k GAUSSIAN_FWHM]
                              [-a KERNEL_SIZE]

source extraction using image segmentation and deblending

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT_FILE, --input-file INPUT_FILE
                        input file name
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
```

```
                        output file name
  -t THRESHOLD, --threshold THRESHOLD
                        detection threshold in sigma (default: 3)
  -u THRESHOLD_FOR_SKY, --threshold-for-sky THRESHOLD_FOR_SKY
                        detection threshold for sky estimate (default: 2)
  -n NPIXELS, --npixels NPIXELS
                        minimum number of pixels for detection (default: 5)
  -s DILATE_SIZE, --dilate-size DILATE_SIZE
                        dilate size (default: 21)
  -m MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 30)
  -r SIGMA_CLIPPING, --sigma-clipping SIGMA_CLIPPING
                        sigma-clipping threshold in sigma (default: 4)
  -k GAUSSIAN_FWHM, --gaussian-fwhm GAUSSIAN_FWHM
                        Gaussian FWHM in pixel for convolution (default: 3)
  -a KERNEL_SIZE, --kernel-size KERNEL_SIZE
                        Gaussian kernel array size in pixel (default: 3)

% ./advobs202202_s16_04.py -i synthetic_0.fits -o debl_0.pdf
% ls -lF debl_0.pdf
-rw-r--r--  1 daisuke  taiwan  21446 May 26 21:49 debl_0.pdf
```

Show the deblended image. Overlapping sources are now separately detected. (Fig. 3)

```
% okular debl_0.pdf
```

5 Producing source catalogue

Make a Python script to produce a catalogue of extracted sources.

Python Code 5: advobs202202_s16_05.py

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

#
# Time-stamp: <2022/05/26 21:58:40 (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.ma

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

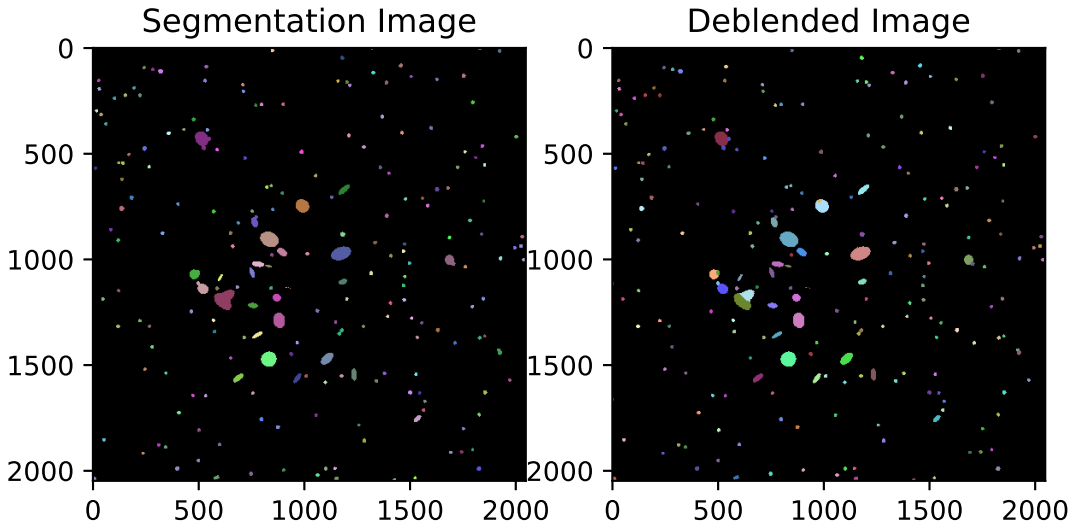

Figure 3: Deblended image produced by photutils package.

```
import astropy.stats
import astropy.visualization
import astropy.visualization.mpl_normalize

# importing photutils module
import photutils.segmentation

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

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

# constructing parser object
desc   = 'source extraction using image segmentation and deblending'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--input-file', default='', \
                     help='input file name')
parser.add_argument ('-o', '--catalogue-file', default='', \
                     help='output catalogue file name')
parser.add_argument ('-t', '--threshold', type=float, default=3.0, \
                     help='detection threshold in sigma (default: 3)')
parser.add_argument ('-u', '--threshold-for-sky', type=float, default=2.0, \
                     help='detection threshold for sky estimate (default: 2)')
parser.add_argument ('-n', '--npixels', type=int, default=5, \
                     help='minimum number of pixels for detection (default: 5)')
parser.add_argument ('-s', '--dilate-size', type=int, default=21, \
                     help='dilate size (default: 21)')
parser.add_argument ('-m', '--maxiters', type=int, default=30, \
                     help='maximum number of iterations (default: 30)')
parser.add_argument ('-r', '--sigma-clipping', type=float, default=4.0, \
                     help='sigma-clipping threshold in sigma (default: 4)')
parser.add_argument ('-k', '--gaussian-fwhm', type=float, default=3.0, \
                     help='Gaussian FWHM in pixel for convolution (default: 3)')
parser.add_argument ('-a', '--kernel-size', type=int, default=3, \
                     help='Gaussian kernel array size in pixel (default: 3)')

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

# file names
file_input     = args.input_file
file_catalogue = args.catalogue_file

# input parameters
threshold         = args.threshold
threshold_for_sky = args.threshold
npixels           = args.npixels
dilate_size       = args.dilate_size
maxiters          = args.maxiters
rejection         = args.sigma_clipping
gaussian_fwhm     = args.gaussian_fwhm
kernel_array_size = args.kernel_size

# making pathlib objects
path_input     = pathlib.Path (file_input)
```

```
path_catalogue = pathlib.Path (file_catalogue)

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

# check of catalogue file name
if (file_catalogue == ''):
    # printing message
    print ("Catalogue file name must be specified.")
    # exit
    sys.exit ()

# existence check of input file
if not (path_input.exists ()):
    # printing message
    print ("ERROR: Input file '%s' does not exist." % (file_input) )
    # exit
    sys.exit ()

# existence check of output file
if (path_catalogue.exists ()):
    # printing message
    print ("ERROR: Output file '%s' exists." % (file_catalogue) )
    # exit
    sys.exit ()

# opening FITS file
with astropy.io.fits.open (file_input) as hdu:
    # reading header and image
    header = hdu[0].header
    image  = hdu[0].data
    # if no image in PrimaryHDU, then read next HDU
    if (header['NAXIS'] == 0):
        header = hdu[1].header
        image  = hdu[1].data

# making source mask
source_mask \
    = photutils.segmentation.make_source_mask (image, threshold_for_sky,
                                               npixels=npixels,
                                               sigclip_iters=maxiters,
                                               dilate_size=dilate_size)

# making masked array
image_masked = numpy.ma.array (image, mask=source_mask)

# sigma-clipping
skybg_mean, skybg_median, skybg_stddev \
    = astropy.stats.sigma_clipped_stats (image, sigma=rejection)

# mode calculation using empirical formula
skybg_mode = 3.0 * skybg_median - 2.0 * skybg_mean

# detection threshold in ADU
threshold_adu = skybg_mode + threshold * skybg_stddev
```

```
# 2D Gaussian kernel for convolution
gaussian_sigma = gaussian_fwhm * astropy.stats.gaussian_fwhm_to_sigma
kernel = astropy.convolution.Gaussian2DKernel (gaussian_sigma, \
                                               x_size=kernel_array_size, \
                                               y_size=kernel_array_size)
kernel.normalize ()

# source detection
image_segm = photutils.segmentation.detect_sources (image, threshold_adu, \
                                                    npixels=npixels, \
                                                    kernel=kernel)

# deblending
image_deblend = photutils.segmentation.deblend_sources (image, image_segm, \
                                                        npixels=npixels, \
                                                        kernel=kernel, \
                                                        nlevels=32, \
                                                        contrast=0.001)

# making a source catalogue
catalogue = photutils.segmentation.SourceCatalog (image, image_deblend)

# making a table
table_source = catalogue.to_table ()

# writing table to a file
astropy.io.ascii.write (table_source, file_catalogue, \
                        format='commented_header')
```

Run the script and generate a catalogue.

```
% chmod a+x advobs202202_s16_05.py
% ./advobs202202_s16_05.py -h
usage: advobs202202_s16_05.py [-h] [-i INPUT_FILE] [-o CATALOGUE_FILE]
                              [-t THRESHOLD] [-u THRESHOLD_FOR_SKY]
                              [-n NPIXELS] [-s DILATE_SIZE] [-m MAXITERS]
                              [-r SIGMA_CLIPPING] [-k GAUSSIAN_FWHM]
                              [-a KERNEL_SIZE]

source extraction using image segmentation and deblending

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT_FILE, --input-file INPUT_FILE
                        input file name
  -o CATALOGUE_FILE, --catalogue-file CATALOGUE_FILE
                        output catalogue file name
  -t THRESHOLD, --threshold THRESHOLD
                        detection threshold in sigma (default: 3)
  -u THRESHOLD_FOR_SKY, --threshold-for-sky THRESHOLD_FOR_SKY
                        detection threshold for sky estimate (default: 2)
  -n NPIXELS, --npixels NPIXELS
                        minimum number of pixels for detection (default: 5)
  -s DILATE_SIZE, --dilate-size DILATE_SIZE
                        dilate size (default: 21)
  -m MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 30)
```

```
  -r SIGMA_CLIPPING, --sigma-clipping SIGMA_CLIPPING
                        sigma-clipping threshold in sigma (default: 4)
  -k GAUSSIAN_FWHM, --gaussian-fwhm GAUSSIAN_FWHM
                        Gaussian FWHM in pixel for convolution (default: 3)
  -a KERNEL_SIZE, --kernel-size KERNEL_SIZE
                        Gaussian kernel array size in pixel (default: 3)

% ./advobs202202_s16_05.py -i synthetic_0.fits -o synthetic_0.cat
% ls -lF synthetic_0.*
-rw-r--r--  1 daisuke  taiwan     53064 May 26 21:58 synthetic_0.cat
-rw-r--r--  1 daisuke  taiwan  33560640 May 26 21:21 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan     26313 May 26 21:20 synthetic_0.log
-rw-r--r--  1 daisuke  taiwan       532 May 26 21:34 synthetic_0.sky
% head -5 synthetic_0.cat
# label xcentroid ycentroid sky_centroid bbox_xmin bbox_xmax bbox_ymin bbox_ymax
 area semimajor_sigma semiminor_sigma orientation eccentricity min_value max_val
ue local_background segment_flux segment_fluxerr kron_flux kron_fluxerr
1 1317.3825433810805 4.209408283381483 None 1309 1326 0 10 162.0 4.1895202214866
37 2.7700761455762275 -2.289684780040178 0.7502169613450786 1038.292039136017 24
64.651595589643 0.0 252070.39274936885 nan 1858445.9170956705 nan
2 1824.4576555461683 4.5164388280826335 None 1814 1835 0 13 257.0 4.344327181660
928 3.2058167482822797 5.104742571076818 0.6748753990013895 1046.6490960385672 9
126.315266910371 0.0 802961.0852690514 nan 1981008.4914666228 nan
3 848.2107907449875 16.34425081570255 None 840 856 8 25 227.0 4.073908301266826
3.8508393817323734 81.90224862885913 0.32636306590357134 1033.148591580933 2392.
7927193983223 0.0 332733.00311524695 nan 3351710.5809855186 nan
4 1919.105403356602 18.332790114519764 None 1911 1928 10 27 257.0 4.191438141837
236 4.074127755282101 65.18651451871641 0.23493161712868785 1009.1585894096195 2
873.476921128506 0.0 405389.03365689376 nan 3762339.7109013814 nan
```

6 Checking locations of detected sources

We now check locations of detected sources to examine whether a source detection is successful.

6.1 Reading a catalogue file

Make a Python script to read table from a file.

Python Code 6: advobs202202_s16_06.py

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

#
# Time-stamp: <2022/05/26 22:03:55 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# import pathlib module
import pathlib

# importing astropy module
import astropy.table
```

```
# constructing parser object
desc   = 'reading table from a file'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-c', '--catalogue-file', default='', \
                     help='input catalogue file name')

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

# catalogue file name
file_catalogue = args.catalogue_file

# making pathlib object
path_catalogue = pathlib.Path (file_catalogue)

# check of catalogue file name
if (file_catalogue == ''):
    # printing message
    print ("ERROR: Catalogue file name must be specified.")
    # exit
    sys.exit ()

# existence check of catalogue file
if not (path_catalogue.exists () ):
    # printing message
    print ("ERROR: Catalogue file '%s' does not exist." % (file_catalogue) )
    # exit
    sys.exit ()

# reading catalogue from a file
table_source = astropy.table.Table.read (file_catalogue, \
                                         format='ascii.commented_header')

# printing table
print (table_source)
```

Run the script.

```
% chmod a+x advobs202202_s16_06.py
% ./advobs202202_s16_06.py -h
usage: advobs202202_s16_06.py [-h] [-c CATALOGUE_FILE]

reading table from a file

optional arguments:
  -h, --help            show this help message and exit
  -c CATALOGUE_FILE, --catalogue-file CATALOGUE_FILE
                        input catalogue file name

% ./advobs202202_s16_06.py -c synthetic_0.cat
label     xcentroid          ycentroid       ...     kron_flux      kron_fluxerr
----- ------------------ ------------------ ... ------------------ ------------
    1 1317.3825433810805  4.209408283381483 ... 1858445.9170956705          nan
    2 1824.4576555461683 4.5164388280826335 ... 1981008.4914666228          nan
    3  848.2107907449875  16.34425081570255 ... 3351710.5809855186          nan
    4  1919.105403356602 18.332790114519764 ... 3762339.7109013814          nan
```

```
     5 1178.4678109781605  50.68188355766648 ...  4750562.253519278          nan
     6   1453.95450769216  86.20443043136027 ...  4593270.149480101          nan
     7  538.5278795066893  92.49174841362752 ...  4752641.732820179          nan
     8 244.12620420573822  96.79820314668643 ...  4249430.900846774          nan
     9 286.48764949283856 105.55758422545497 ...  4419666.378261308          nan
    10 322.05746139496347 113.28400412247878 ... 4154857.8062972906          nan
    11 1324.0154751198318 120.45334190406852 ...  4424412.268979271          nan
    12 1748.9159861304258 135.54906755563118 ...  5024798.806317508          nan
    13 1501.6048712637926  139.0097113524377 ...   4103908.73108532          nan
    14 1622.5777220898758  151.8426953843813 ...  4511974.984576078          nan
    15 31.127014775185565 158.22394537965596 ...  4081376.327418643          nan
    16   532.540139493534 158.19658383528218 ... 4851917.6656156145          nan
    17 1157.5503225392845  159.9151735730691 ...  4200225.539127865          nan
    18 1199.8305877144346  164.2072333125622 ...  4480985.714452844          nan
    19 1385.4638590618315 167.71214467816796 ...  5003677.266695118          nan
    20  709.8693516284611  177.8926579311269 ...  4586930.302883558          nan
    21 1213.2779398010196 179.48924386157162 ...  4383580.965428051          nan
    22  197.6370651359818 192.61361426633448 ...  4289278.593143646          nan
    23  34.18690115874715   196.631792762364 ...  4459286.496873904          nan
    24  712.6423990684393 208.35249105124555 ...  4051126.354190624          nan
    25 156.56713865070355 226.57452337351498 ...  4712094.836384056          nan
    26 225.08300938544454 227.54772724651792 ...  4655315.117982717          nan
    27 105.06578552072766 240.46670194506845 ... 4101347.0730903395          nan
    28 1792.4845213519761 261.00950908563686 ...   4660927.49652862          nan
    29  798.7707471720221   271.001045752844 ...  4499243.551212331          nan
    30  900.3538874790553  271.0357574588114 ...   4708194.68841584          nan
    31  771.0410091477873 275.25906928366584 ...  4409268.305456835          nan
    32  21.50094824067474 300.23224314012924 ...  3598100.060770761          nan
...                   ...                  ... ...                ...          ...
   199 1100.2656967806015   1989.48798248358 ...  4869575.874124884          nan
   200 1143.3017543663095 1988.6113055611927 ...  4203429.842915859          nan
   201  902.4526550434771 1995.4026125766916 ...  4329641.583351096          nan
   202  747.9106930080364  2013.034250588871 ...  4795473.279445667          nan
   203  1061.877372883849  2019.270281840161 ...  4419715.575680645          nan
   204 1520.6829339248143 2032.3774692808033 ...  3000429.559832855          nan
   205 1151.2816609896106  2036.502841557685 ... 2875325.9331853734          nan
   206 10.675288303176485  2036.159082363054 ... 1856449.9515112222          nan
   207    1423.7541747267  2041.459425618785 ... 2067072.7105373698          nan
   208  515.9073366049707 431.91376457080366 ... 56279581.757370524          nan
   209  552.3072757203756 432.75556029247014 ...  5226444.549875462          nan
   210  526.3288871507076 474.86190563796794 ...  5025511.692557634          nan
   211  980.4004139753441   726.014061086177 ...  7142586.774906763          nan
   212   990.576381802252  750.8390614195602 ...  46574068.90974154          nan
   213  772.5586725533457   803.022161615212 ... 5130899.4625446955          nan
   214   764.837767452321  827.6143652817341 ...  21404423.19480129          nan
   215  1682.044458594508 1003.4582758690885 ...  26949445.64468032          nan
   216 1701.1238998248573 1022.1310012141452 ...  5235218.583515472          nan
   217 498.54262890613865 1064.9982132543876 ...  5222948.436495615          nan
   218  478.7789570853484 1073.2379141784274 ... 24749989.049638063          nan
   219 499.70329391007493 1113.6696904634266 ...  4150978.355842525          nan
   220  522.1110467190521 1141.3849419911808 ... 32967945.002106003          nan
   221  917.8956492363232 1133.9994330477043 ...  68057.05231007529          nan
   222  923.3004830243578 1135.0012570462973 ... 63308.143247847875          nan
   223  638.2078500999799 1171.9658886442062 ...  40808772.52016239          nan
   224  614.0637883279638 1200.8431933856725 ...  60929270.13727881          nan
   225 1176.3189968465183 1336.2653065426937 ...  4827553.130557649          nan
   226 1164.3799174163578   1351.91634247508 ...  4947188.350935448          nan
   227 1913.8305611630035 1427.9734588875979 ... 3390142.3712668233          nan
   228 1919.4967963446452  1441.970868789391 ...  4280333.050276688          nan
```

```
  229  590.2525299513516 2028.5422431919933 ... 3578056.6979320007          nan
  230  578.2078028455209 2033.5545237828255 ...  2660861.837898192          nan
Length = 230 rows
```

6.2 Marking locations of detected sources

Make a Python script to mark locations of detected sources.

Python Code 7: advobs202202_s16_07.py

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

#
# Time-stamp: <2022/05/26 22:18:43 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing astropy module
import astropy.table
import astropy.visualization

# importing photutils module
import photutils.aperture

# importing matplotlib module
import matplotlib.pyplot

# constructing parser object
desc   = 'reading table from a file'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-c', '--catalogue-file', default='', \
                     help='input catalogue file name')
parser.add_argument ('-i', '--input-file', default='', \
                     help='input FITS file name')
parser.add_argument ('-o', '--output-file', default='', \
                     help='output file name')
parser.add_argument ('-r', '--radius', type=float, default=10.0, \
                     help='radius of aperture in pixel (default: 10)')

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

# catalogue file name
file_catalogue = args.catalogue_file
file_input     = args.input_file
file_output    = args.output_file
radius         = args.radius

# making pathlib objects
```

```
path_input     = pathlib.Path (file_input)
path_catalogue = pathlib.Path (file_catalogue)
path_output    = pathlib.Path (file_output)

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

# check of catalogue file name
if (file_catalogue == ''):
    # printing message
    print ("Catalogue file name must be specified.")
    # exit
    sys.exit ()

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

# existence check of input file
if not (path_input.exists ()):
    # printing message
    print ("ERROR: Input file '%s' does not exist." % (file_input) )
    # exit
    sys.exit ()

# existence check of catalogue file
if not (path_catalogue.exists () ):
    # printing message
    print ("ERROR: Catalogue file '%s' does not exist." % (file_catalogue) )
    # exit
    sys.exit ()

# existence check of output file
if (path_output.exists ()):
    # printing message
    print ("ERROR: Output file '%s' exists." % (file_output) )
    # exit
    sys.exit ()

# reading catalogue from a file
table_source = astropy.table.Table.read (file_catalogue, \
                                         format='ascii.commented_header')

# positions of detected sources
list_x = list (table_source['xcentroid'])
list_y = list (table_source['ycentroid'])

# opening FITS file
with astropy.io.fits.open (file_input) as hdu:
    # reading header and image
    header = hdu[0].header
```

```
    image  = hdu[0].data
    # if no image in PrimaryHDU, then read next HDU
    if (header['NAXIS'] == 0):
        header = hdu[1].header
        image  = hdu[1].data


# 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]')

# normalisation
norm \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image) )

# plotting image
im = ax.imshow (image, origin='upper', cmap='viridis', norm=norm)

# plotting marks
for i in range ( len (list_x) ):
    # making a circle to indicate location of detected source
    source = matplotlib.patches.Circle (xy=(list_x[i], list_y[i]), \
                                        radius=30, \
                                        fill=False, color="red", \
                                        linewidth=1)
    # plotting location of detected source
    ax.add_patch (source)

# saving file
fig.savefig (file_output, dpi=225)
```

Execute the script, and check the image. (Fig. 4)

```
% chmod a+x advobs202202_s16_07.py
% ./advobs202202_s16_07.py -h
usage: advobs202202_s16_07.py [-h] [-c CATALOGUE_FILE] [-i INPUT_FILE]
                              [-o OUTPUT_FILE] [-r RADIUS]

reading table from a file

optional arguments:
  -h, --help            show this help message and exit
  -c CATALOGUE_FILE, --catalogue-file CATALOGUE_FILE
                        input catalogue file name
  -i INPUT_FILE, --input-file INPUT_FILE
                        input FITS file name
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        output file name
  -r RADIUS, --radius RADIUS
                        radius of aperture in pixel (default: 10)

% ./advobs202202_s16_07.py -i synthetic_0.fits -c synthetic_0.cat -o srcs_0.pdf
```

```
% ls -lF srcs_0.pdf
-rw-r--r--  1 daisuke  taiwan  650272 May 26 22:20 srcs_0.pdf
% okular sources.pdf
```

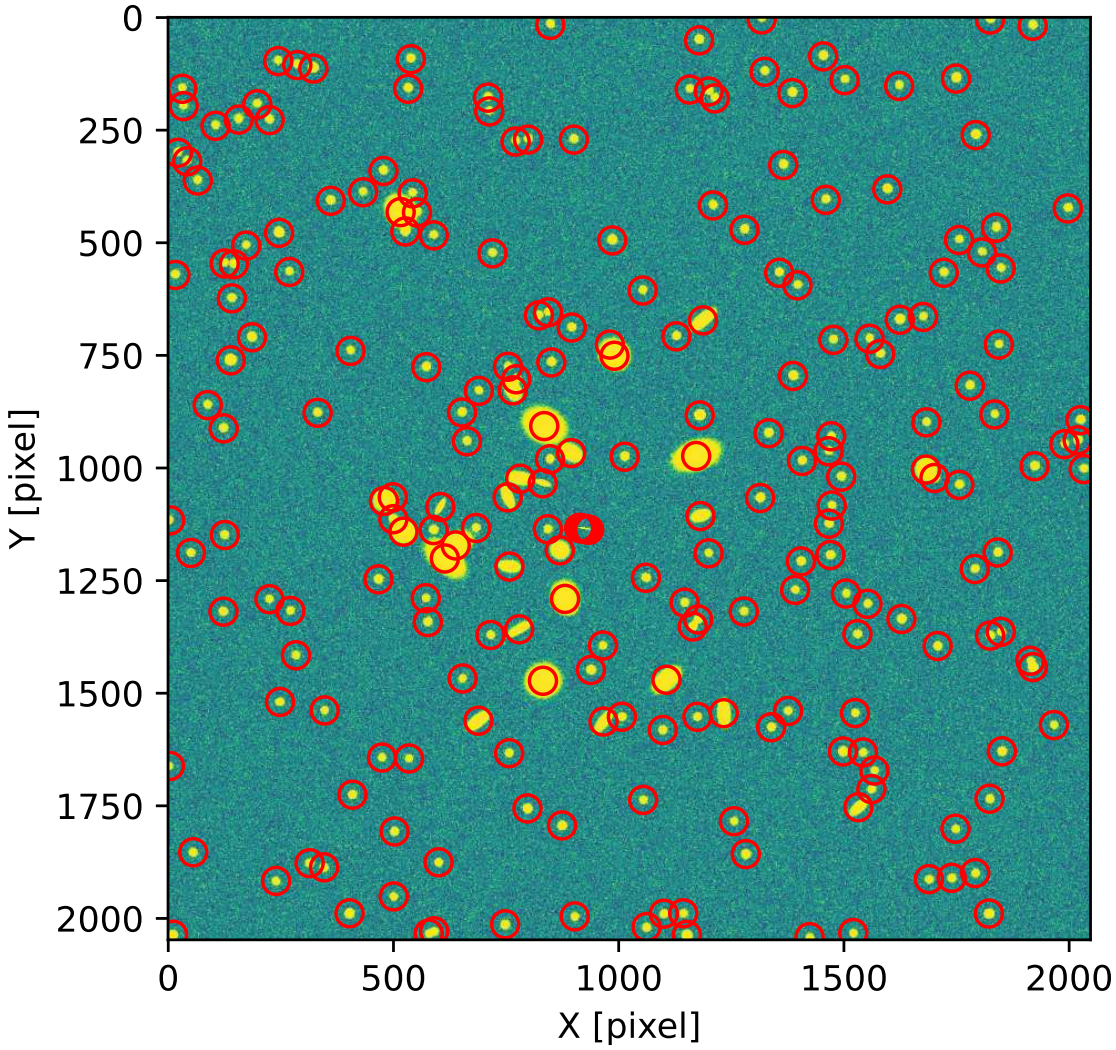

Figure 4: Locations of detected sources.

7 Source extraction of DSS image

Try source extraction of DSS image.

7.1 Downloading DSS image

Download a DSS image.

7.2 Source extraction of DSS image

Carry out source extraction of DSS image. Try image segmentation and de-blending. Make an output file containing positions of extracted sources. Check locations of detected sources.

8 For your further reading

Read followings.

1. "Data Tables" of Astropy
   - `https://docs.astropy.org/en/stable/table/`
2. "FITS File Handling" of Astropy
   - `https://docs.astropy.org/en/stable/io/fits/`
3. "Visualization" of Astropy
   - `https://docs.astropy.org/en/stable/visualization/`
4. "Random sampling" of Numpy
   - `https://numpy.org/doc/stable/reference/random/`
5. "Datasets" of photutils
   - `https://photutils.readthedocs.io/en/stable/datasets.html`
6. "Image Segmentation" of photutils
   - `https://photutils.readthedocs.io/en/stable/segmentation.html`

9 Exercises

1. What is image segmentation? Describe about it.
2. What is deblending? Describe about it.
3. What is convolution? Describe about it.
4. Make your own Python script to carry out source extraction for one of g'-band images taken on 14 February 2021. Show all the Python scripts you made. Describe what you have done. Show the result of source extraction. Are sources successfully detected?
5. Choose one r'-band image taken on 14 February 2021, and apply source extraction using your own Python script. Discuss the result of source extraction.
6. Choose one i'-band image taken on 14 February 2021, and apply source extraction using your own Python script. Discuss the result of source extraction.
7. Use `astroquery` package to download a SDSS image. Apply source extraction using your own Python script to the SDSS image you have downloaded. Discuss the result of source extraction.
8. Choose a source extraction software (e.g. IRAF's daofind, source extractor). Install it on your computer. Learn about the usage of the software. Apply the software to one synthetic image and one real image. Use `photutils` to carry out source extraction of same images. Compare results and give a discussion.