Advanced Astronomical Observations 2021
Session 18: 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 2021 to June 2021.
  - 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 Python scripts for this session

All the Python scripts needed for this session are available at github.com. Visit the web page `https://github.com/kinoshitadaisuke/ncu_advobs_202102` to view files. (Fig. 1)

You may try following command to download all the Python scripts for this session.

```
% git clone https://github.com/kinoshitadaisuke/ncu_advobs_202102.git
Cloning into 'ncu_advobs_202102'...
remote: Enumerating objects: 147, done.
remote: Counting objects: 100% (147/147), done.
remote: Compressing objects: 100% (142/142), done.
Receiving objects: 100% (147/147), 91.13 KiB | 740.00 KiB/s, done.
remote: Total 147 (delta 73), reused 0 (delta 0), pack-reused 0
Resolving deltas: 100% (73/73), done.
% ls
ncu_advobs_202102/
% ls -l ncu_advobs_202102/
total 1
-rw-r--r--  1 daisuke  wheel  35149 May 26 00:41 LICENSE
-rw-r--r--  1 daisuke  wheel    568 May 26 00:41 README
-rw-r--r--  1 daisuke  wheel    343 May 26 00:41 REQUIREMENTS
-rw-r--r--  1 daisuke  wheel    342 May 26 00:41 download_python.csh
-rw-r--r--  1 daisuke  wheel    495 May 26 00:41 download_stds.csh
drwxr-xr-x  2 daisuke  wheel    512 May 26 00:41 s16_skybackground/
```

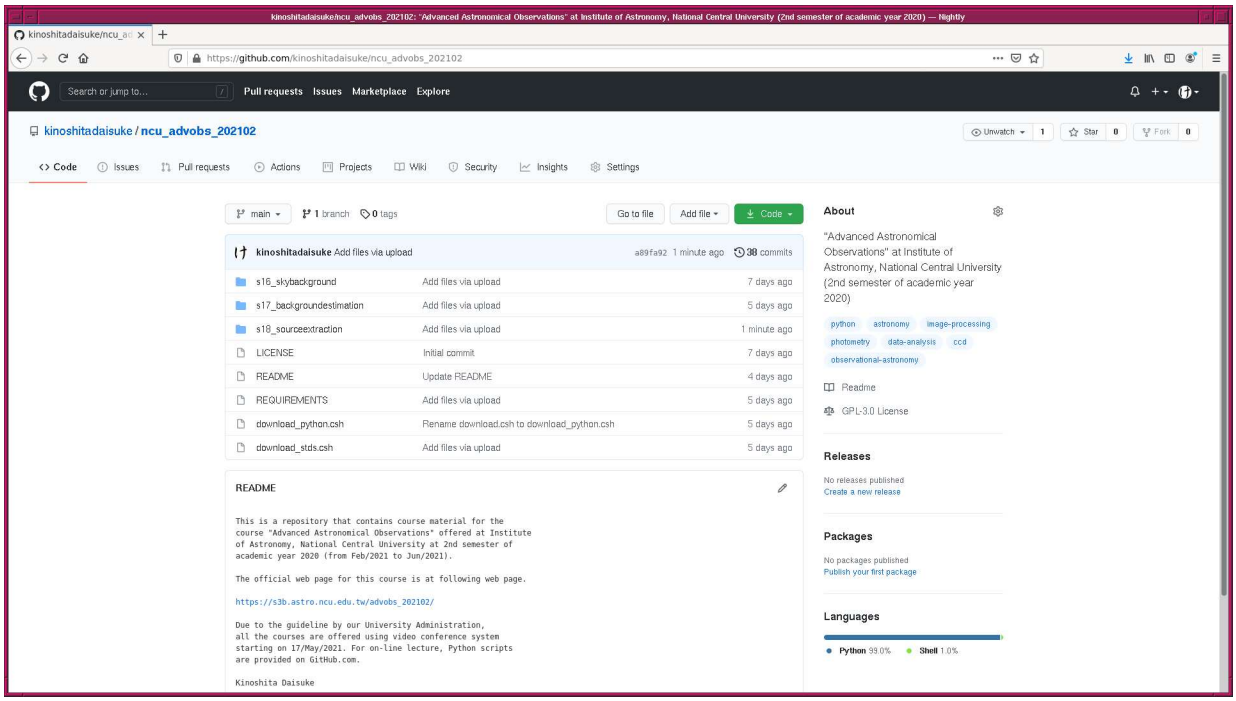

Figure 1: The GitHub repository for Python scripts for this course.

```
drwxr-xr-x  2 daisuke  wheel    512 May 26 00:41 s17_backgroundestimation/
drwxr-xr-x  2 daisuke  wheel    512 May 26 00:41 s18_sourceextraction/
% ls -l ncu_advobs_202102/s18_sourceextraction/
total 1
-rw-r--r--  1 daisuke  wheel     74 May 26 00:41 README
-rw-r--r--  1 daisuke  wheel   8676 May 26 00:41 ao2021_s18_01.py
-rw-r--r--  1 daisuke  wheel   3485 May 26 00:41 ao2021_s18_02.py
-rw-r--r--  1 daisuke  wheel   5223 May 26 00:41 ao2021_s18_03.py
-rw-r--r--  1 daisuke  wheel   5541 May 26 00:41 ao2021_s18_04.py
-rw-r--r--  1 daisuke  wheel   5093 May 26 00:41 ao2021_s18_05.py
-rw-r--r--  1 daisuke  wheel    919 May 26 00:41 ao2021_s18_06.py
-rw-r--r--  1 daisuke  wheel   2793 May 26 00:41 ao2021_s18_07.py
% head -15 ncu_advobs_202102/s18_sourceextraction/ao2021_s18_01.py
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2021/05/25 19:52:05 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

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

If above command fails on your computer, check whether or not you have the command `git`. You see something like following if you have the command `git` on your computer.

```
% which git
/usr/pkg/bin/git
```

You see following if you do not have the command `git` installed on your computer.

```
% which git
git: Command not found.
```

If you do not have the command `git`, install `git` on your computer.

2 Generating synthetic data

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

Python Code 1: ao2021_s18_01.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# 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

# check of output file name
if not (file_output[-5:] == '.fits'):
    print ("Output file must be a FITS file.")
    sys.exit ()

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

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

# generating random numbers for stars
```

```
position_x = numpy.random.default_rng().uniform(0, image_size_x, nstars)
position_y = numpy.random.default_rng().uniform(0, image_size_y, nstars)
theta_deg  = numpy.random.default_rng().uniform(0, 360, nstars)
psf_x      = numpy.random.default_rng().normal(loc=fwhm_x, \
                                               scale=fwhm_stddev_x, \
                                               size=nstars)
psf_y      = numpy.random.default_rng().normal(loc=fwhm_y, \
                                               scale=fwhm_stddev_y, \
                                               size=nstars)
powerlaw   = numpy.random.default_rng().power(1.5, size=nstars)
flux       = flux_min / powerlaw

# generating random numbers for galaxies
centre_gal_x   = numpy.random.default_rng().uniform(image_size_x * 0.3, \
                                                    image_size_x * 0.7)
centre_gal_y   = numpy.random.default_rng().uniform(image_size_y * 0.3, \
                                                    image_size_y * 0.7)
position_gal_x = numpy.random.default_rng().normal(loc=centre_gal_x, \
                                                   scale=300, size=ngals)
position_gal_y = numpy.random.default_rng().normal(loc=centre_gal_y, \
                                                   scale=300, size=ngals)
theta_gal_deg  = numpy.random.default_rng().uniform(0, 360, ngals)
psf_gal_x      = numpy.random.default_rng().normal(loc=fwhm_gal_x, \
                                                   scale=fwhm_gal_stddev_x, \
                                                   size=ngals)
psf_gal_y      = numpy.random.default_rng().normal(loc=fwhm_gal_y, \
                                                   scale=fwhm_gal_stddev_y, \
                                                   size=ngals)
powerlaw_gal   = numpy.random.default_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 ao2021_s18_01.py
% ./ao2021_s18_01.py -h
usage: ao2021_s18_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]

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

% ./ao2021_s18_01.py -n 200 -g 30 -l synthetic_0.log -o synthetic_0.fits
% ls -l synthetic_0.*
-rw-r--r--  1 daisuke  taiwan  33560640 May 25 19:52 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan     26323 May 25 19:52 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
```

```
8286.899718308106 1131.8238486040154 1512.3437697817265 3.6227715718041065 3.546
356935359455 2.8808131879196752
```

Use Ginga to check the image. (Fig. 2)

```
% ls -l *.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 25 19:32 synthetic_0.fits
% ginga ginga synthetic_0.fits &
```

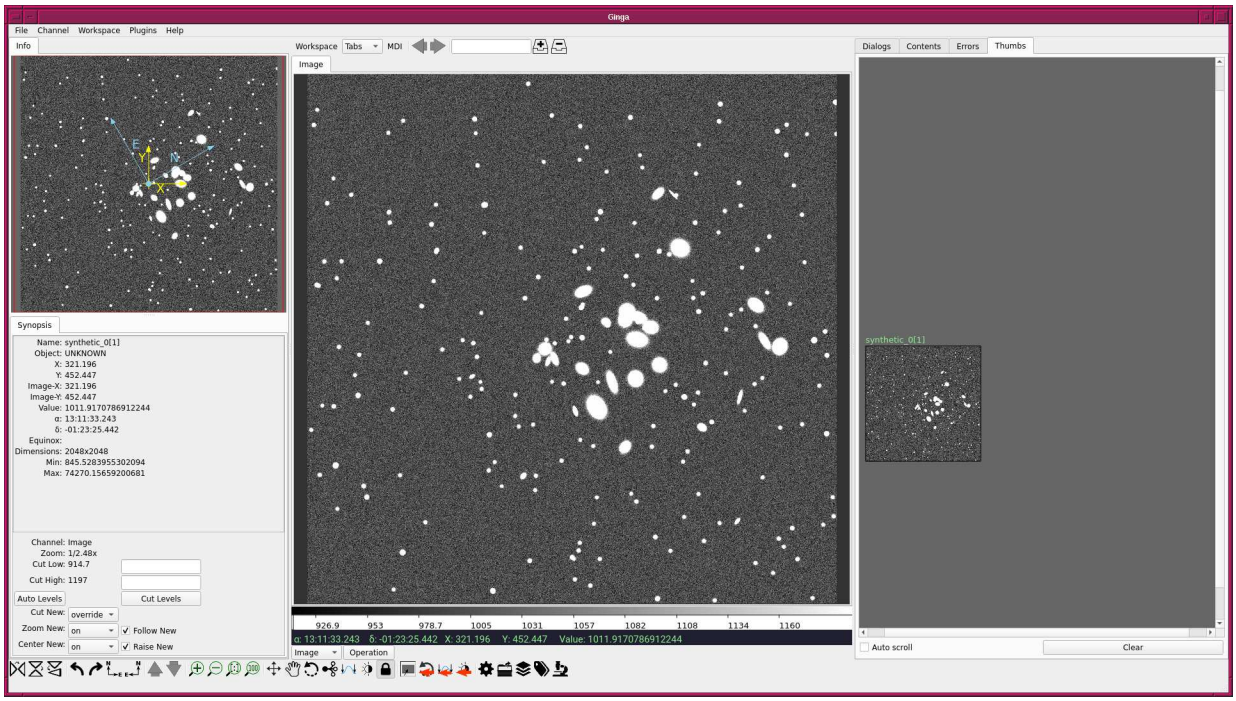

Figure 2: A synthetic image of artificial stars and galaxies.

3 Background estimation

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

Python Code 2: ao2021_s18_02.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# 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

# check of input file name
if not (file_input[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    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 ao2021_s18_02.py
% ./ao2021_s18_02.py -i synthetic_0.fits > synthetic_0.sky
% ls -l synthetic_0.sky
-rw-r--r--  1 daisuke  taiwan  533 May 25 20:15 synthetic_0.sky
% cat synthetic_0.sky
#
# background estimation using source detection and masking
#
# date/time = 2021-05-25 20:15:04.071345
#
# 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
#
1000.806396 1000.544645 1000.021144 30.698029
```

Estimated sky background level is 1000.02 ADU, and it is very close to expected value of 1000 ADU.

4 Source detection using image segmentation

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

Python Code 3: ao2021_s18_03.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# 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

# check of input file name
if not (file_input[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# check of output file name
if not ( (file_output[-4:] == '.eps') or (file_output[-4:] == '.pdf') \
         or (file_output[-4:] == '.png') or (file_output[-3:] == '.ps')):
    print ("Output file must be either EPS, PDF, PNG, or PS.")
    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, \
                                                    filter_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 ao2021_s18_03.py
% ./ao2021_s18_03.py -h
usage: ao2021_s18_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)

% ./ao2021_s18_03.py -i synthetic_0.fits -o segm_0.pdf
% ls -l segm_0.pdf
-rw-r--r--  1 daisuke  taiwan  552347 May 25 22:17 segm_0.pdf
```

Show the segmentation image. Sources are successfully detected. (Fig. 3)

```
% xpdf segm_0.pdf
```

5 Deblending

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

Python Code 4: ao2021_s18_04.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# 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
```

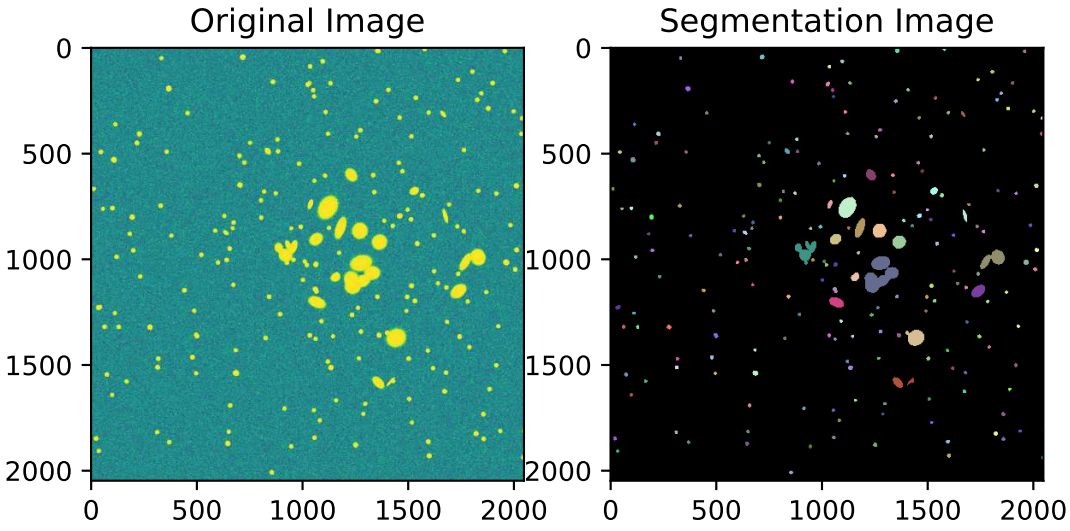

Figure 3: Segmantation image produced by photutils package.

```
# 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

# check of input file name
if not (file_input[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# check of output file name
```

```
if not ( (file_output[-4:] == '.eps') or (file_output[-4:] == '.pdf') \
         or (file_output[-4:] == '.png') or (file_output[-3:] == '.ps')):
    print ("Output file must be either EPS, PDF, PNG, or PS.")
    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, \
                                                    filter_kernel=kernel)

# deblending
image_deblend = photutils.segmentation.deblend_sources (image, image_segm, \
                                                        npixels=npixels, \
                                                        filter_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 ao2021_s18_04.py
% ./ao2021_s18_04.py -h
usage: ao2021_s18_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)

% ./ao2021_s18_04.py -i synthetic_0.fits -o debl_0.pdf
% ls -l debl_0.pdf
-rw-r--r--  1 daisuke  taiwan  26525 May 25 22:41 debl_0.pdf
```

Show the deblended image. Overlapping sources are now separately detected. (Fig. 4)

```
% xpdf debl_0.pdf
```

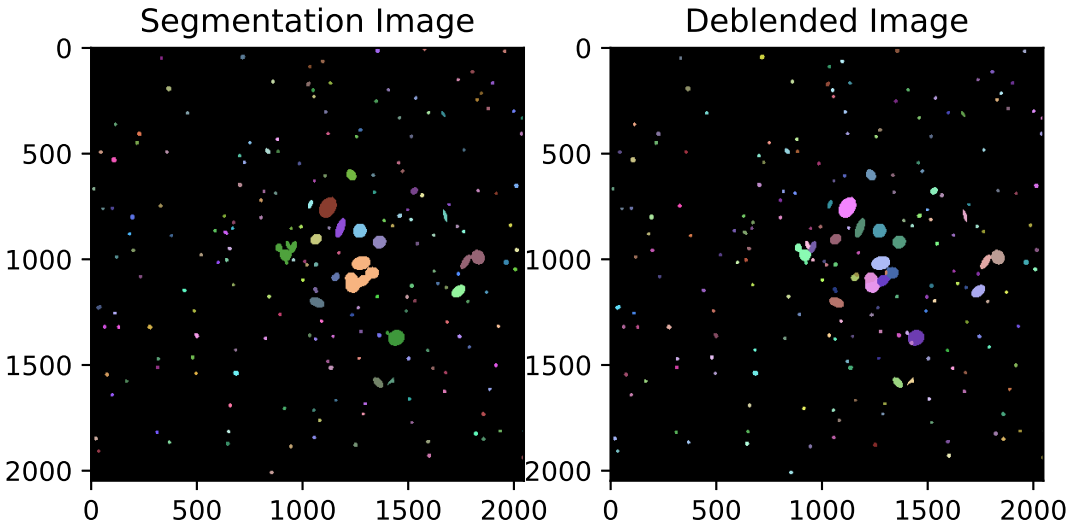

Figure 4: Deblended image produced by photutils package.

6 Producing source catalogue

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

Python Code 5: ao2021_s18_05.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# 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 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

# check of input file name
if not (file_input[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# check of catalogue file name
if (file_catalogue == ''):
    print ("Catalogue file name must be specified.")
    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, \
                                                    filter_kernel=kernel)

# deblending
image_deblend = photutils.segmentation.deblend_sources (image, image_segm, \
                                                        npixels=npixels, \
                                                        filter_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 ao2021_s18_05.py
% ./ao2021_s18_05.py -h
usage: ao2021_s18_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)

% ./ao2021_s18_05.py -i synthetic_0.fits -o synthetic_0.cat
% ls -l synthetic_0.*
-rw-r--r--  1 daisuke  taiwan     50585 May 25 23:29 synthetic_0.cat
-rw-r--r--  1 daisuke  taiwan  33560640 May 25 19:52 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan     26323 May 25 19:52 synthetic_0.log
-rw-r--r--  1 daisuke  taiwan       533 May 25 20:15 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 1574.0978521704335 9.982982295413304 None 1566 1582 2 18 230.0 3.9753478909053
235 3.8458274706516424 34.16175107628468 0.2531803486138813 1049.9978087355921 2
873.2449555491366 0.0 361331.1280431652 nan 2726705.3613020903 nan
2 1352.9768954217009 16.941187872075307 None 1342 1363 6 28 387.0 4.268090109147
736 4.079621961012992 46.64777865890545 0.2938793917451648 1015.8213778885365 13
145.972021459185 0.0 1306692.2372832568 nan 3346211.8342881147 nan
3 1952.4678849129932 19.22332838540533 None 1944 1961 11 27 232.0 4.031233073188
095 3.8735467619111867 24.890662567320287 0.2769516633031454 1039.5346651674975
2707.374690480418 0.0 355850.4045260383 nan 3667741.2643187675 nan
4 715.7481588942989 46.96622473148286 None 705 727 36 58 423.0 4.018354549987698
 3.8888335255400794 -36.079653357501826 0.25184475860732985 1047.1185682738485 2
5857.38362504063 0.0 2234579.1482860697 nan 3773820.977226943 nan
```

7 Checking locations of detected sources

7.1 Reading a catalogue file

Make a Python script to read table from a file.

Python Code 6: ao2021_s18_06.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# 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

# check of catalogue file name
if (file_catalogue == ''):
    print ("Catalogue file name must be specified.")
    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 ao2021_s18_06.py
% ./ao2021_s18_06.py -h
usage: ao2021_s18_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

% ./ao2021_s18_06.py -c synthetic_0.cat
label     xcentroid          ycentroid       ...     kron_flux      kron_fluxerr
----- ------------------ ------------------ ... ------------------ ------------
    1 1574.0978521704335  9.982982295413304 ... 2726705.3613020903          nan
    2 1352.9768954217009 16.941187872075307 ... 3346211.8342881147          nan
    3 1952.4678849129932  19.22332838540533 ... 3667741.2643187675          nan
    4  715.7481588942989  46.96622473148286 ...  3773820.977226943          nan
    5 331.93755558517444 51.830583350059705 ... 4076467.9558321363          nan
    6  1092.986742009767  67.44149613905944 ...  4734296.742780196          nan
    7 1033.4146467119951  91.32364437591251 ...  4510685.265908936          nan
  ...                ...                ... ...                ...          ...
  213  1159.987502160512  1074.009291832633 ... 5238823.2545966245          nan
  214 1151.8284250768675 1088.7875245900357 ... 15327754.867012516          nan
  215 1398.3144606685642 1347.8882468945296 ...  5396441.713792821          nan
  216 1441.1703616007803 1370.7150630871936 ...  77336671.78884366          nan
  217 1420.2871519444136 1395.3597221078344 ...  7477875.261236841          nan
  218 1413.6171789748205 1578.4791538753993 ...  6553306.858807882          nan
  219  1426.194282898542 1577.0944759141312 ... 3035876.5677858614          nan
Length = 219 rows
```

7.2 Marking locations of detected sources

Make a Python script to mark locations of detected sources.

Python Code 7: ao2021_s18_07.py

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

# importing argparse module
import argparse
```

```
# importing sys module
import sys

# 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

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

# check of input FITS file name
if not (file_input[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# check of output file name
if not ( (file_output[-4:] == '.eps') or (file_output[-4:] == '.pdf') \
         or (file_output[-4:] == '.png') or (file_output[-3:] == '.ps')):
    print ("Output file must be either EPS, PDF, PNG, or PS.")
    sys.exit ()

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

# positions of apertures
list_x = list (table_source['xcentroid'])
```

```
list_y = list (table_source['ycentroid'])
positions = []
for i in range ( len (list_x) ):
    positions.append ( (list_x[i], list_y[i]) )

# apertures
apertures = photutils.aperture.CircularAperture (positions, r=radius)

# 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

# matplotlib
norm \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image) )
matplotlib.pyplot.imshow (image, origin='upper', cmap='viridis', norm=norm)
apertures.plot (color='red', lw=1.0, alpha=0.5)
matplotlib.pyplot.title ("Detected sources")
matplotlib.pyplot.savefig (file_output, dpi=225)
```

Execute the script, and check the image. (Fig. 5)

```
% chmod a+x ao2021_s18_07.py
% ./ao2021_s18_07.py -h
usage: ao2021_s18_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)

% ./ao2021_s18_07.py -r 25 -i synthetic_0.fits -c synthetic_0.cat -o sources.pdf
% ls -l sources.pdf
-rw-r--r--  1 daisuke  taiwan  1670403 May 26 00:15 sources.pdf
% xpdf sources.pdf
```

8 For your training

Read followings.

1. “Data Tables” of Astropy

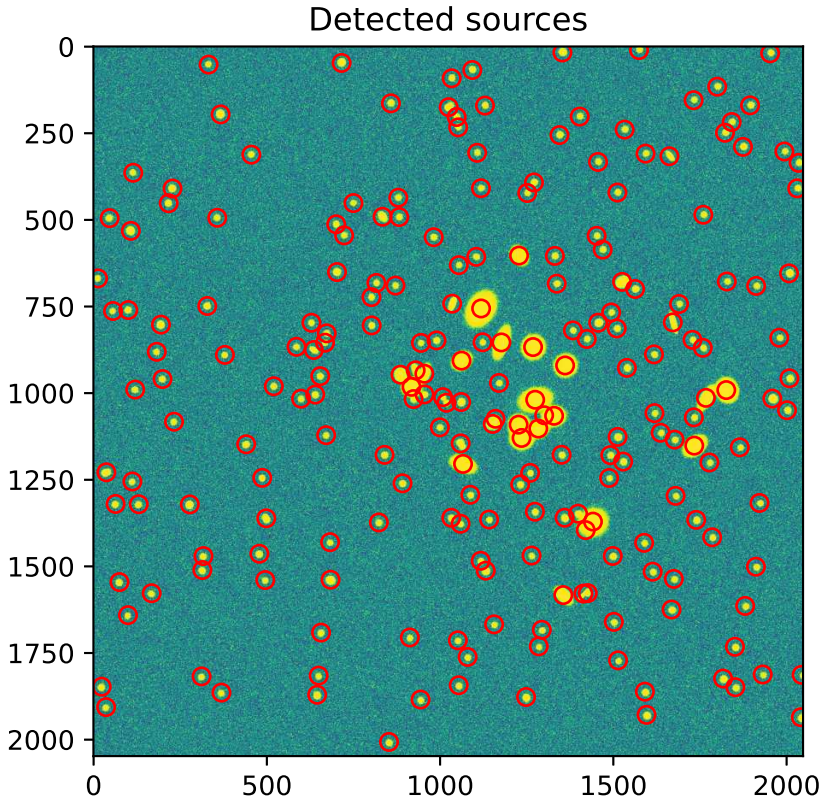

Figure 5: Locations of detected sources.

   - `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 Assignment

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.