Advanced Astronomical Observations 2022
Session 15: Estimation of Background Level

Kinoshita Daisuke

20 May 2022
publicly accessible version

About this file...

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

For this session, we try to estimate background level.

1 Generating synthetic data

We first generate a synthetic image to estimate background level.

1.1 Preparing an Astropy table

For making a synthetic image with stars, we need to make a Astropy table. Make a Python script to generate an Astropy table.

Python Code 1: advobs202202_s15_01.py

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

#
# Time-stamp: <2022/05/19 16:50:54 (CST) daisuke>
#

# importing numpy module
import numpy
import numpy.random

# importing astropy module
import astropy.table
```

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

# number of stars to generate
nstars = 100

# flux of faintest stars
flux_min = 1000.0

# image size
image_size_x = 2048
image_size_y = 2048

# FWHM of PSF
fwhm_x        = 3.5
fwhm_y        = 3.5
fwhm_stddev_x = 0.1
fwhm_stddev_y = 0.1

# generating random numbers
rng        = numpy.random.default_rng ()
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

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

# adding data to the table
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

# printing table
print ("#")
print ("# Astropy table")
print ("#")
print (table_stars)

# printing table information
print ("#")
print ("# Astropy table information")
print ("#")
print (table_stars.info)
```

Execute the script.

```
% chmod a+x advobs202202_s15_01.py
% ./advobs202202_s15_01.py
#
# Astropy table
```

```
#
    amplitude            x_mean        ...         theta
------------------ ------------------- ... -------------------
1225.9080840908794  1445.277565347674 ...    4.83478946271626
3531.2161191534233 1844.3181437977125 ...  1.5393492510804578
1421.0832079295376 1049.1009843794857 ...  1.1002094111167748
 1686.887270409156 1142.4505079913567 ...   5.151188498526226
   1061.1224083335 1547.1185950269626 ...   5.010082531143072
1190.8178439441158  696.8449605442665 ...   5.312457124548242
1604.5321565390784  89.75252677065828 ...   2.733277963979946
               ...                 ... ...                 ...
1156.1937244799635 1252.6225994906545 ...   4.475220236981059
1404.5209361374489 2021.5459603321572 ...  3.2100864862319423
1178.0994675487725  718.5121649380269 ... 0.17828784165390413
1189.9884046215404 1291.1399434622936 ...   3.943684744392084
1770.8130385785216   555.689894447517 ... 0.27983673117757146
1153.7002096253807  1359.555640353468 ...  0.4651777919772129
2391.8319344953193 1658.3814794533896 ...  1.9739058042766935
Length = 100 rows
#
# Astropy table information
#
<Table length=100>
   name    dtype
--------- -------
amplitude float64
   x_mean float64
   y_mean float64
 x_stddev float64
 y_stddev float64
    theta float64
```

1.2 Generating a synthetic image with 200 artificial stars

Make a Python script to generate a synthetic image with 200 artificial stars.

Python Code 2: advobs202202_s15_02.py

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

#
# Time-stamp: <2022/05/19 17:33:52 (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
import numpy.random
```

```
# importing astropy module
import astropy.table

# importing photutils module
import photutils.datasets

# constructing parser object
desc   = 'generating a synthetic image with artificial stars'
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 ('-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 ('-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')

# 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 to generate
nstars = args.nstars

# 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

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

# output file name
file_output = args.output_file

# making pathlib object
```

```
path_output = pathlib.Path (file_output)

# 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 ()

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

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

# command name
command = sys.argv[0]

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

# generating random numbers
rng        = numpy.random.default_rng ()
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

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

# adding data to the table
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

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

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

# generating synthetic image (stars + sky background)
image = image_stars + image_sky
```

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

# making FITS header
header = wcs.to_header ()

# adding comments to the header
header['history'] = "FITS file created by the command \"%s\"" % (command)
header['history'] = "Updated on %s" % (now)
header['comment'] = "image size = %d x %d" % (image_size_x, image_size_y)
header['comment'] = "number of stars = %d" % (nstars)
header['comment'] = "flux_min = %f" % (flux_min)
header['comment'] = "fwhm_x = %f" % (fwhm_x)
header['comment'] = "fwhm_y = %f" % (fwhm_y)
header['comment'] = "fwhm_stddev_x = %f" % (fwhm_stddev_x)
header['comment'] = "fwhm_stddev_y = %f" % (fwhm_stddev_y)
header['comment'] = "sky_mean = %f" % (sky_mean)
header['comment'] = "sky_stddev = %f" % (sky_stddev)
header['comment'] = "file_output = %s" % (file_output)

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

Run the script, and generate a synthetic image.

```
% chmod a+x advobs202202_s15_02.py
% ./advobs202202_s15_02.py -h
usage: advobs202202_s15_02.py [-h] [-x SIZE_X] [-y SIZE_Y] [-n NSTARS]
                              [-f FLUX_MIN] [-p FWHM_PSF] [-d FWHM_STDDEV]
                              [-s SKY] [-e SKY_STDDEV] [-o OUTPUT_FILE]

generating a synthetic image with artificial stars

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)
  -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)
  -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

% ./advobs202202_s15_02.py -n 200 -o synthetic_0.fits
% ls -lF *.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 17:38 synthetic_0.fits
```

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

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

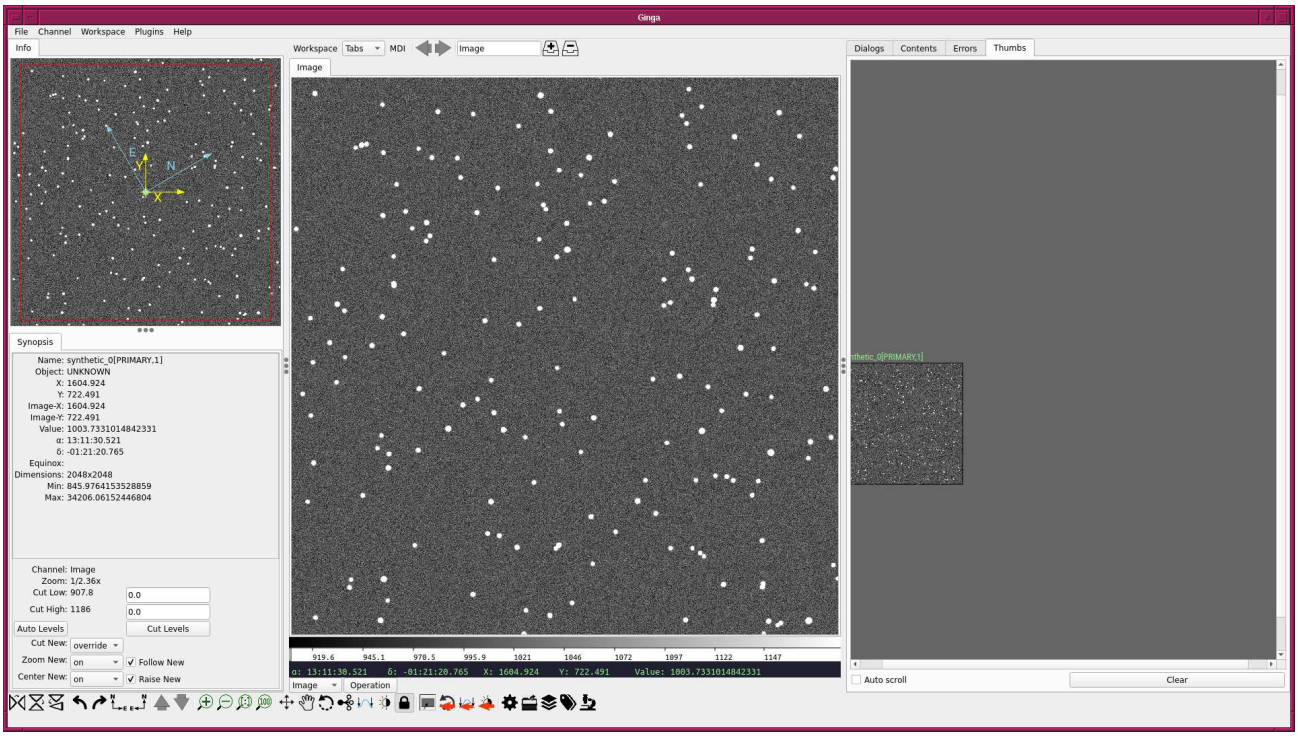

Figure 1: The image of synthetic data generated by `advobs202202_s15_02.py`.

1.3 Generating a synthetic image with 200 artificial stars and 30 galaxies

Make a Python script to generate a synthetic image with 200 artificial stars and 30 galaxies.

Python Code 3: advobs202202_s15_03.py

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

#
# Time-stamp: <2022/05/19 21:02:23 (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
import numpy.random
```

```
# importing astropy module
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')

# 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
file_output = args.output_file

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

# 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 ()

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

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

# command name
command = sys.argv[0]

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

# generating random numbers for stars
rng        = numpy.random.default_rng ()
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
position_gal_x = rng.normal (loc=image_size_x / 2.0, scale=300, size=ngals)
position_gal_y = rng.normal (loc=image_size_y / 2.0, 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

# 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 FITS header
header = wcs.to_header ()

# adding comments to the header
header['history'] = "FITS file created by the command \"%s\"" % (command)
header['history'] = "Updated on %s" % (now)
header['comment'] = "image size = %d x %d" % (image_size_x, image_size_y)
header['comment'] = "number of stars = %d" % (nstars)
header['comment'] = "flux_min = %f" % (flux_min)
header['comment'] = "fwhm_x = %f" % (fwhm_x)
header['comment'] = "fwhm_y = %f" % (fwhm_y)
header['comment'] = "fwhm_stddev_x = %f" % (fwhm_stddev_x)
header['comment'] = "fwhm_stddev_y = %f" % (fwhm_stddev_y)
header['comment'] = "number of galaxies = %d" % (ngals)
header['comment'] = "fwhm_gal_x = %f" % (fwhm_gal_x)
header['comment'] = "fwhm_gal_y = %f" % (fwhm_gal_y)
header['comment'] = "fwhm_gal_stddev_x = %f" % (fwhm_gal_stddev_x)
header['comment'] = "fwhm_gal_stddev_y = %f" % (fwhm_gal_stddev_y)
header['comment'] = "sky_mean = %f" % (sky_mean)
header['comment'] = "sky_stddev = %f" % (sky_stddev)
header['comment'] = "file_output = %s" % (file_output)

# writing a FITS file
```

```
astropy.io.fits.writeto (file_output, image, header=header)
```

Execute the script, and generate a FITS file.

```
% chmod a+x advobs202202_s15_03.py
% ./advobs202202_s15_03.py -h
usage: advobs202202_s15_03.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

% ./advobs202202_s15_03.py -n 200 -g 30 -o synthetic_1.fits
% ls -lF *.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 17:38 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 21:07 synthetic_1.fits
```

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

```
% ginga synthetic_1.fits &
```

2 Detecting and masking sources

Make a Python script to detect and mask sources on the image.

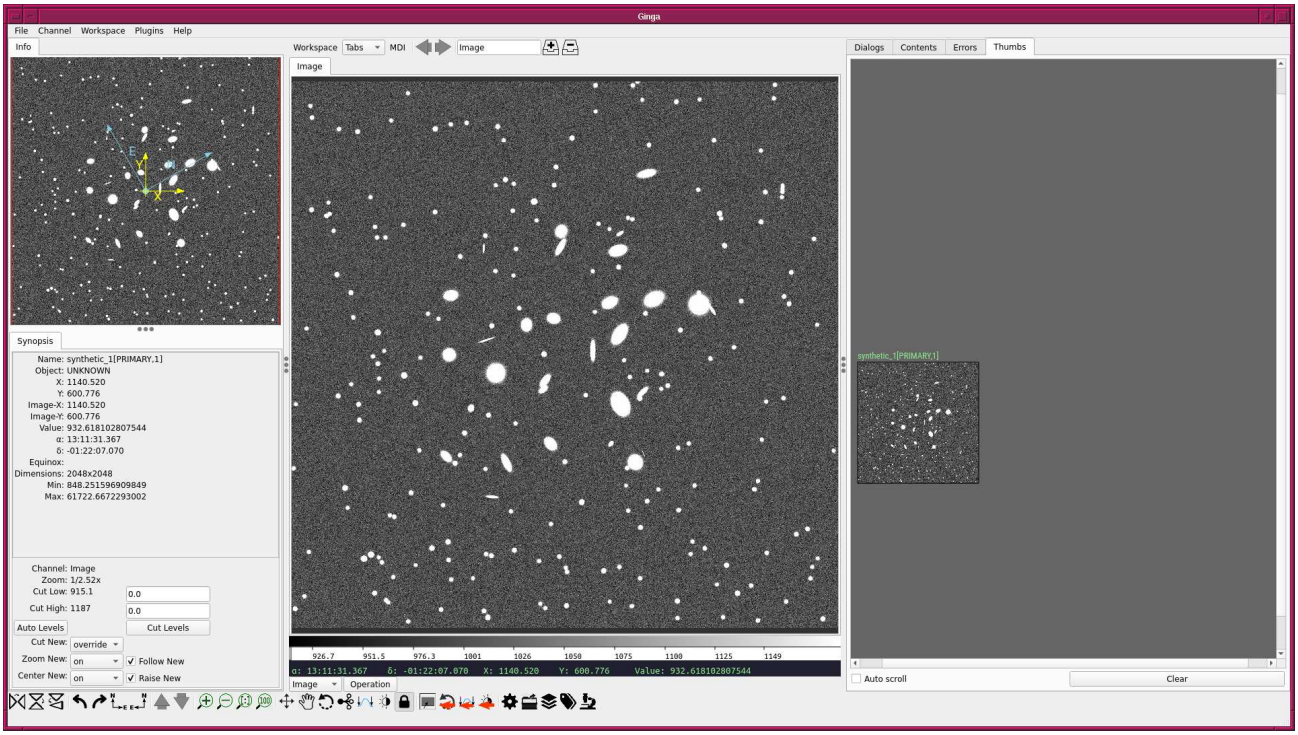

Figure 2: A synthetic image with 200 stars and 30 galaxies.

Python Code 4: advobs202202_s15_04.py

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

#
# Time-stamp: <2022/05/19 21:24:51 (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
import numpy.ma

# importing astropy module
import astropy.io.fits

# importing photutils module
import photutils.segmentation

# constructing parser object
desc    = 'detecting and masking sources'
```

```
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=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)')

# 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
npixels     = args.npixels
dilate_size = args.dilate_size
maxiters    = args.maxiters

# 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 == '.fits'):
    # printing message
    print ("ERROR: Output 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 does not exist!")
    # exit
    sys.exit ()

# now
datetime_now = datetime.datetime.now ()

# command name
command = sys.argv[0]
```

```
# existence check of output file
if (path_output.exists ()):
    # printing message
    print ("ERROR: output file exists!")
    # 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, \
                                               sigclip_iters=maxiters, \
                                               dilate_size=dilate_size)

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

# adding comments in header
header['history'] = "FITS file created by the command \"%s\"" % (command)
header['comment'] = "Updated on %s" % (datetime_now)
header['comment'] = "Detecting and masking sources"
header['comment'] = "Original file = %s" % (file_input)
header['comment'] = "Options:"
header['comment'] = "  threshold   = %f sigma" % (threshold)
header['comment'] = "  npixels     = %d pixels" % (npixels)
header['comment'] = "  dilate_size = %d pixels" % (dilate_size)
header['comment'] = "  maxiters    = %d" % (maxiters)

# writing a FITS file
astropy.io.fits.writeto (file_output, \
                         numpy.ma.filled (image_masked, fill_value=0.0), \
                         header=header)
```

Run the script.

```
% chmod a+x advobs202202_s15_04.py
% ./advobs202202_s15_04.py -h
usage: advobs202202_s15_04.py [-h] [-i INPUT_FILE] [-o OUTPUT_FILE]
                              [-t THRESHOLD] [-n NPIXELS] [-s DILATE_SIZE]
                              [-m MAXITERS]

detecting and masking sources

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: 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)

% ./advobs202202_s15_04.py -i synthetic_1.fits -o synthetic_1_masked.fits
% ls -lF *.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 17:38 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 21:07 synthetic_1.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 21:54 synthetic_1_masked.fits
```

Use Ginga to display the image. (Fig. 3)

```
% ginga synthetic_1_masked.fits &
```

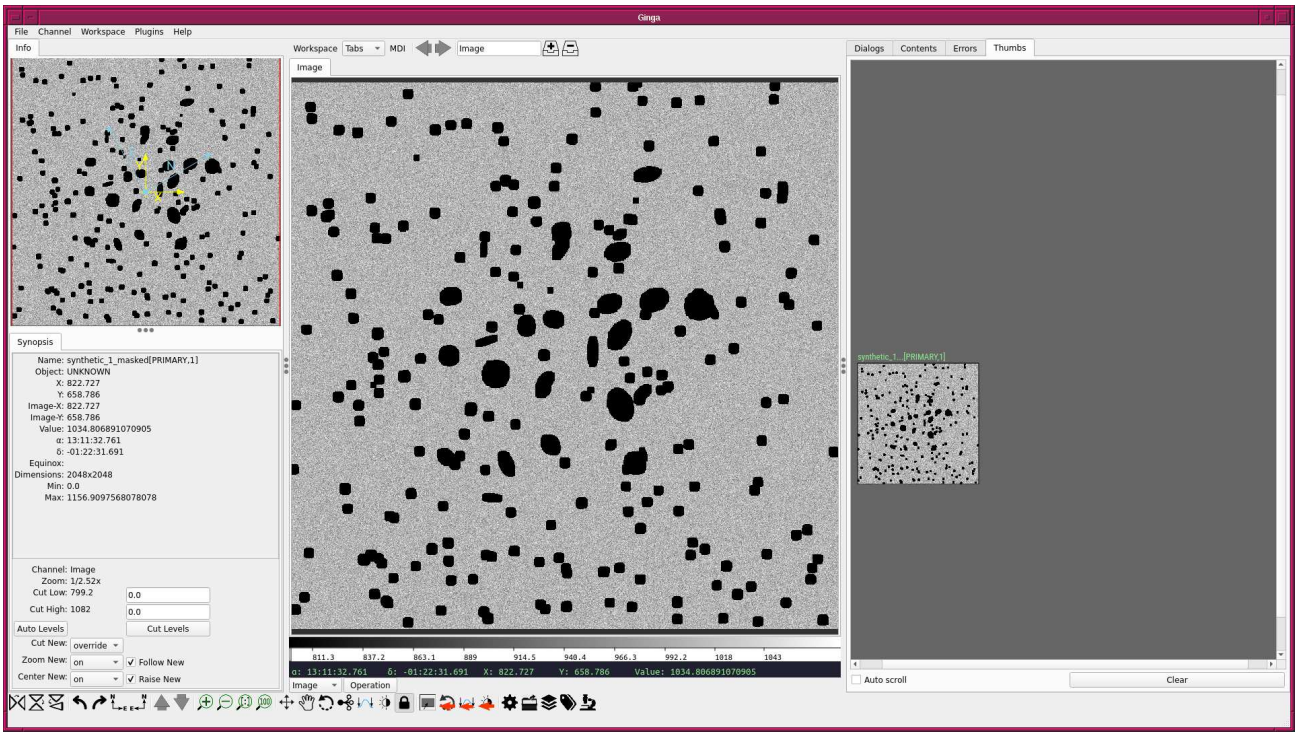

Figure 3: Masked image. Stars and galaxies are successfully masked.

3 Background estimation

Make a Python script to estimate background level.

Python Code 5: advobs202202_s15_05.py

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

```
#
# Time-stamp: <2022/05/19 22:05:05 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# import 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

# constructing parser object
desc   = 'estimating sky background level'
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 does not exist!")
    # 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, \
                                               sigclip_iters=maxiters, \
                                               dilate_size=dilate_size)

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

# calculating mean, median, and stddev using 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 ("# file name = %s" % (file_input) )
print ("#")
print ("mean   = %f ADU" % skybg_mean)
print ("median = %f ADU" % skybg_median)
print ("mode   = %f ADU" % skybg_mode)
print ("stddev = %f ADU" % skybg_stddev)
```

Execute the script.

```
% chmod a+x advobs202202_s15_05.py
% ./advobs202202_s15_05.py -h
usage: advobs202202_s15_05.py [-h] [-i INPUT_FILE] [-t THRESHOLD] [-n NPIXELS]
                              [-s DILATE_SIZE] [-m MAXITERS]
                              [-r SIGMA_CLIPPING]
```

```
estimating sky background level

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_s15_05.py -i synthetic_0.fits
#
# file name = synthetic_0.fits
#
mean   = 1000.417968 ADU
median = 1000.294323 ADU
mode   = 1000.047032 ADU
stddev = 30.339068 ADU
% ./advobs202202_s15_05.py -i synthetic_1.fits
#
# file name = synthetic_1.fits
#
mean   = 1000.776938 ADU
median = 1000.487817 ADU
mode   = 999.909575 ADU
stddev = 30.669613 ADU
```

The mode seems to be a good estimator of the sky background level.

4 Dealing with real data

4.1 Copying a reduced image

If you have finished CCD data reduction of the data taken on 14/Feb/2021, then copy one FITS file to currently working directory.

```
% cp -pi ../../s09/script_09/lot_20210214r/lot_20210214_0185_df.fits .
% ls -lF *.fits
-rw-r--r--  1 daisuke  taiwan  33566400 May 11  2021 lot_20210214_0185_df.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 17:38 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 21:07 synthetic_1.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 21:54 synthetic_1_masked.fits
```

If you have not yet finished CCD data reduction of the data taken on 14/Feb/2021, then download the file.

```
% curl -k -o lot_20210214_0185_df.fits \
? https://s3b.astro.ncu.edu.tw/advobs_202202/data/lot_20210214_0185_df.fits
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 32.0M  100 32.0M    0     0  5720k      0  0:00:05  0:00:05 --:--:-- 5741k
```

```
% ls -lF *.fits
-rw-r--r--  1 daisuke  taiwan  33566400 May 19 22:15 lot_20210214_0185_df.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 17:38 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 21:07 synthetic_1.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 21:54 synthetic_1_masked.fits
```

4.2 Visualising image using Ginga

Use Ginga to visualise image. (4)

```
% ginga lot_20210214_0185_df.fits &
```

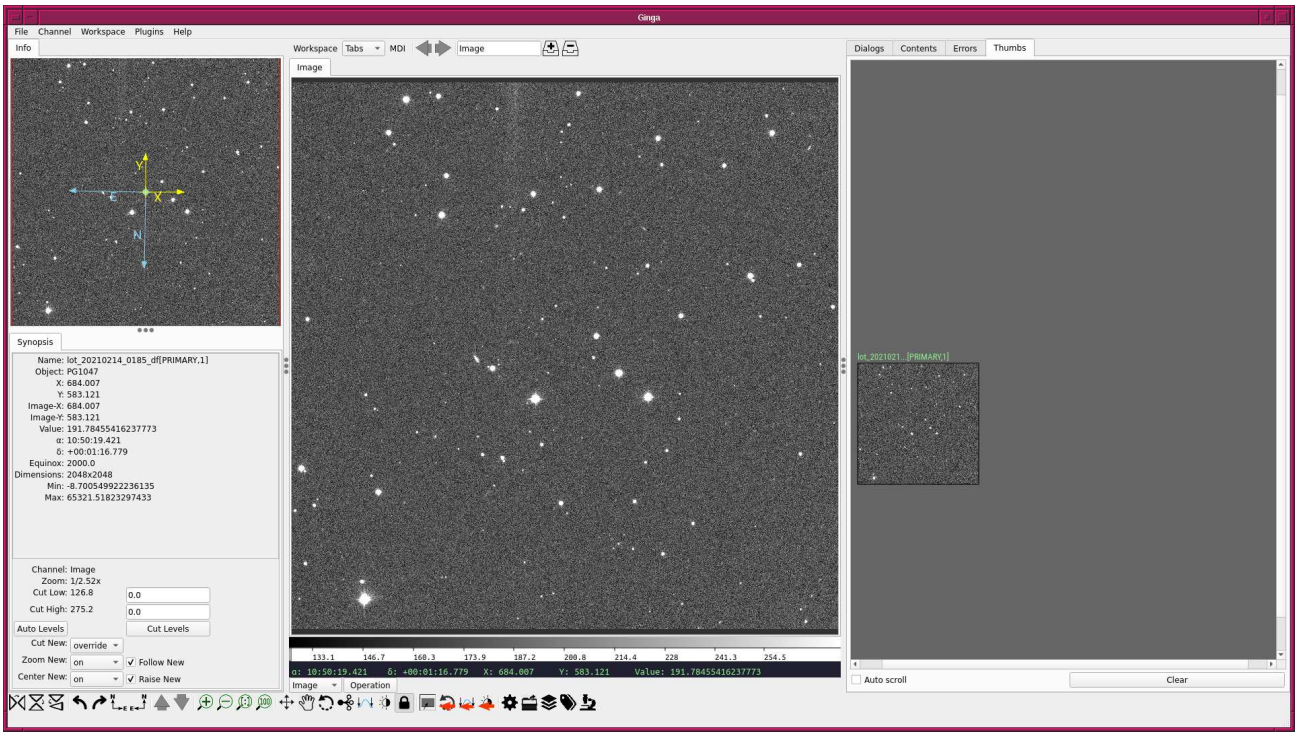

Figure 4: The image of `lot_20210214_0185_df.fits`.

4.3 Detecting stars and masking them

Detect and mask stars.

```
% ./advobs202202_s15_04.py -i lot_20210214_0185_df.fits -o 0185_masked.fits
% ls -lF *.fits
-rw-r--r--  1 daisuke  taiwan  33569280 May 19 22:17 0185_masked.fits
-rw-r--r--  1 daisuke  taiwan  33566400 May 19 22:15 lot_20210214_0185_df.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 17:38 synthetic_0.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 21:07 synthetic_1.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 19 21:54 synthetic_1_masked.fits
```

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

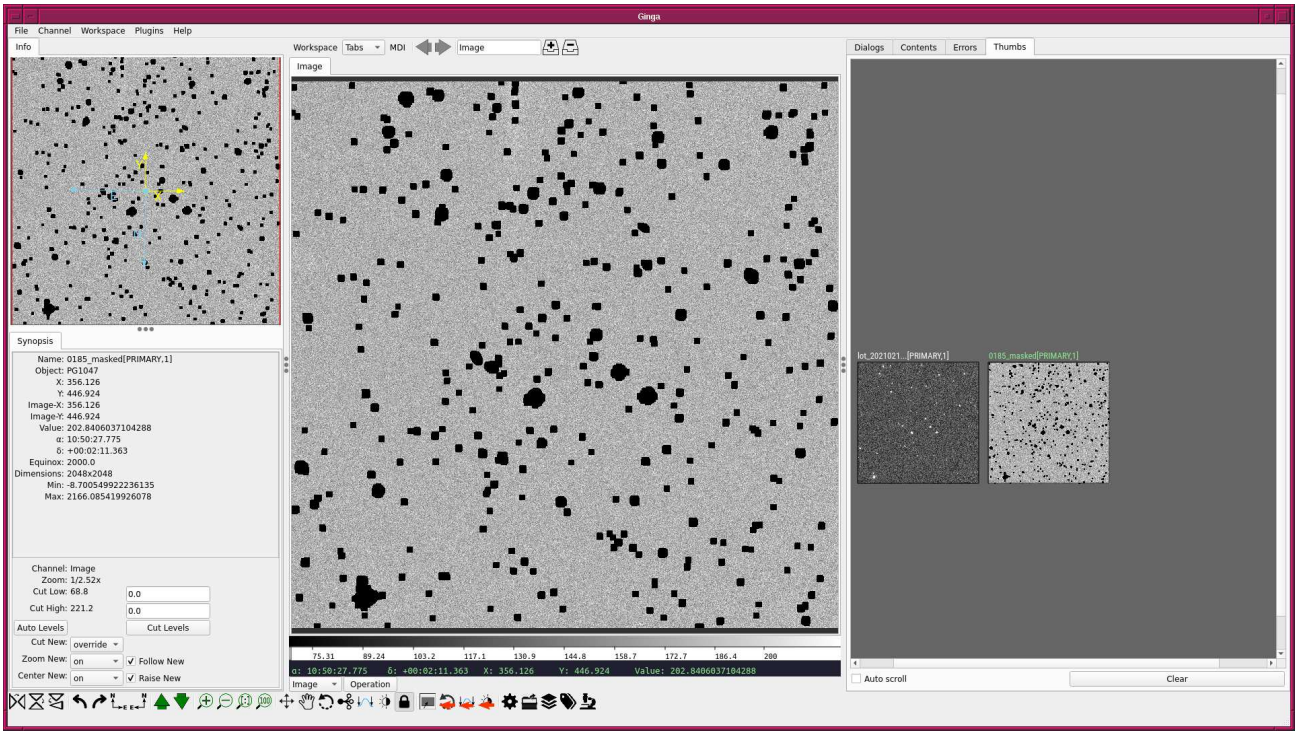

Figure 5: Masked image of `lot_20210214_0185_df.fits`.

4.4 Estimating sky background level

Estimate the sky background level.

```
% ./advobs202202_s15_05.py -i lot_20210214_0185_df.fits
#
# file name = lot_20210214_0185_df.fits
#
mean   = 175.566294 ADU
median = 175.347725 ADU
mode   = 174.910588 ADU
stddev = 16.413741 ADU
```

5 For your further reading

Read followings.

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

6 Exercise

None.