Advanced Astronomical Observations 2022
Session 17: Image Alignment

Kinoshita Daisuke

10 June 2022
publicly accessible version

For this session, we try image alignment.

1 Scikit-Image package

For this session, we need a package named `scikit-image`.

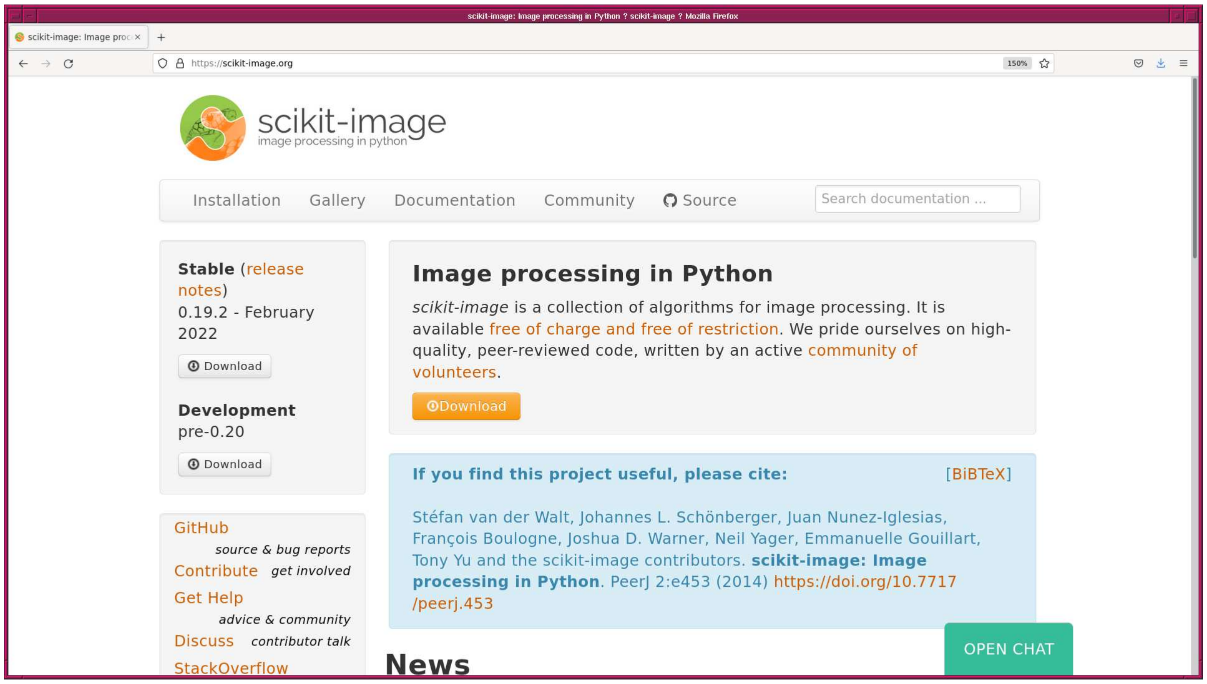
- scikit-image
  - `https://scikit-image.org/` (Fig. 1)

Try following to check whether or not you have `scikit-image` installed on your computer. If the package is properly installed on your computer, then you can successfully import `scikit-image`. (Fig. 2)

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

If you do not have `scikit-image` installed on your computer, then you see a message like below.

```
% python3.9
Python 3.9.12 (main, Apr  6 2022, 23:48:29)
[GCC 10.3.0] on netbsd9
```

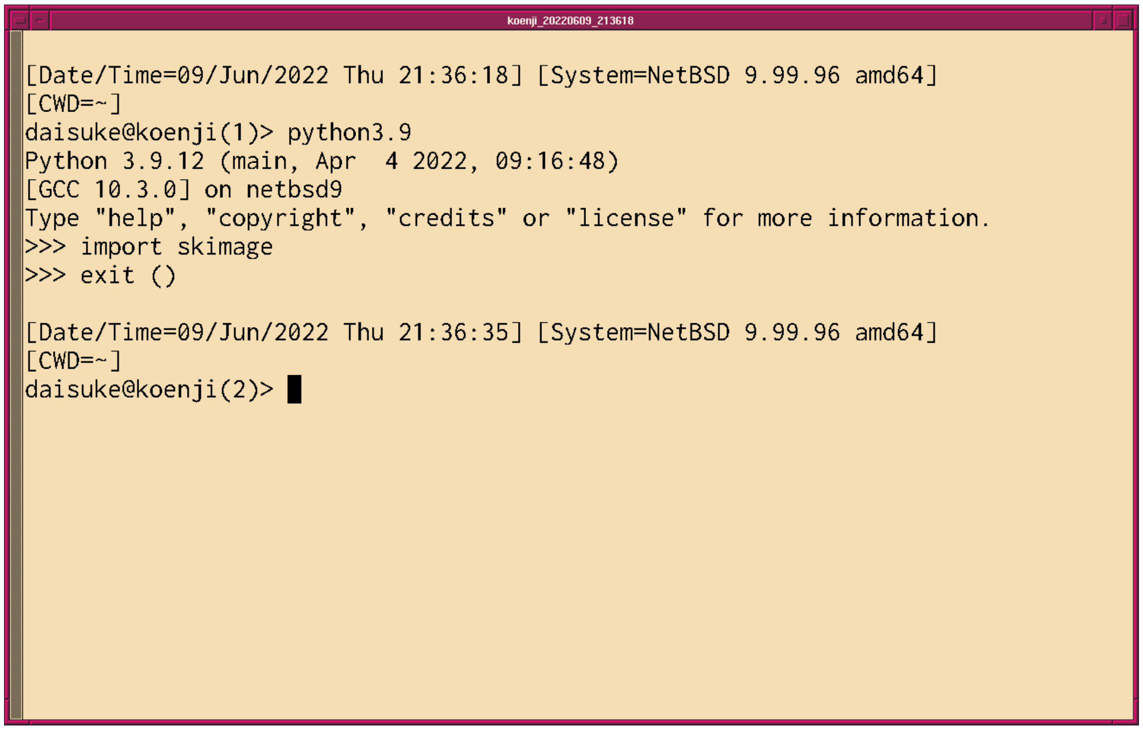


scikit-image is a collection of algorithms for image processing. It is available free of charge and free of restriction. We pride ourselves on high-quality, peer-reviewed code, written by an active community of volunteers.

If you find this project useful, please cite:

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors. **scikit-image: Image processing in Python**. PeerJ 2:e453 (2014) https://doi.org/10.7717/peerj.453

Figure 1: The official web page of scikit-image package.



```
[Date/Time=09/Jun/2022 Thu 21:36:18] [System=NetBSD 9.99.96 amd64]
[CWD=~]
daisuke@koenji(1)> python3.9
Python 3.9.12 (main, Apr  4 2022, 09:16:48)
[GCC 10.3.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import skimage
>>> exit ()

[Date/Time=09/Jun/2022 Thu 21:36:35] [System=NetBSD 9.99.96 amd64]
[CWD=~]
daisuke@koenji(2)>
```

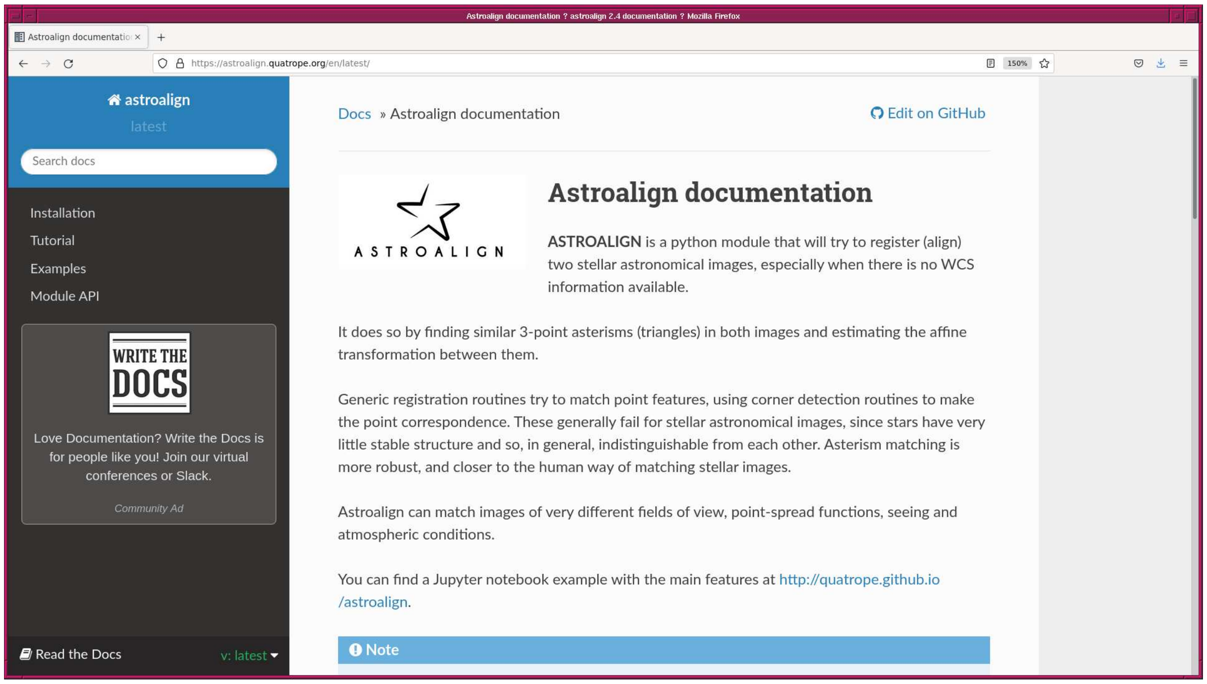
Figure 2: Importing scikit-image package.

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import skimage
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'skimage'
>>> exit ()
```

2 Astroalign package

For this session, we need a package named **`astroalign`**.

- astroalign
  - `https://astroalign.readthedocs.io/` (Fig. 3)
  - `https://pypi.org/project/astroalign/`


Astroalign documentation ? astroalign 2.4 documentation ? Mozilla Firefox
Astroalign documentatio ×
https://astroalign.quatrope.org/en/latest/
150%
astroalign
latest
Search docs
Installation
Tutorial
Examples
Module API
WRITE THE DOCS
Love Documentation? Write the Docs is for people like you! Join our virtual conferences or Slack.
Community Ad
Read the Docs
v: latest
Docs » Astroalign documentation
Edit on GitHub
ASTROALIGN
Note

Astroalign documentation

ASTROALIGN is a python module that will try to register (align) two stellar astronomical images, especially when there is no WCS information available.

It does so by finding similar 3-point asterisms (triangles) in both images and estimating the affine transformation between them.

Generic registration routines try to match point features, using corner detection routines to make the point correspondence. These generally fail for stellar astronomical images, since stars have very little stable structure and so, in general, indistinguishable from each other. Asterism matching is more robust, and closer to the human way of matching stellar images.

Astroalign can match images of very different fields of view, point-spread functions, seeing and atmospheric conditions.

You can find a Jupyter notebook example with the main features at http://quatrope.github.io/astroalign.

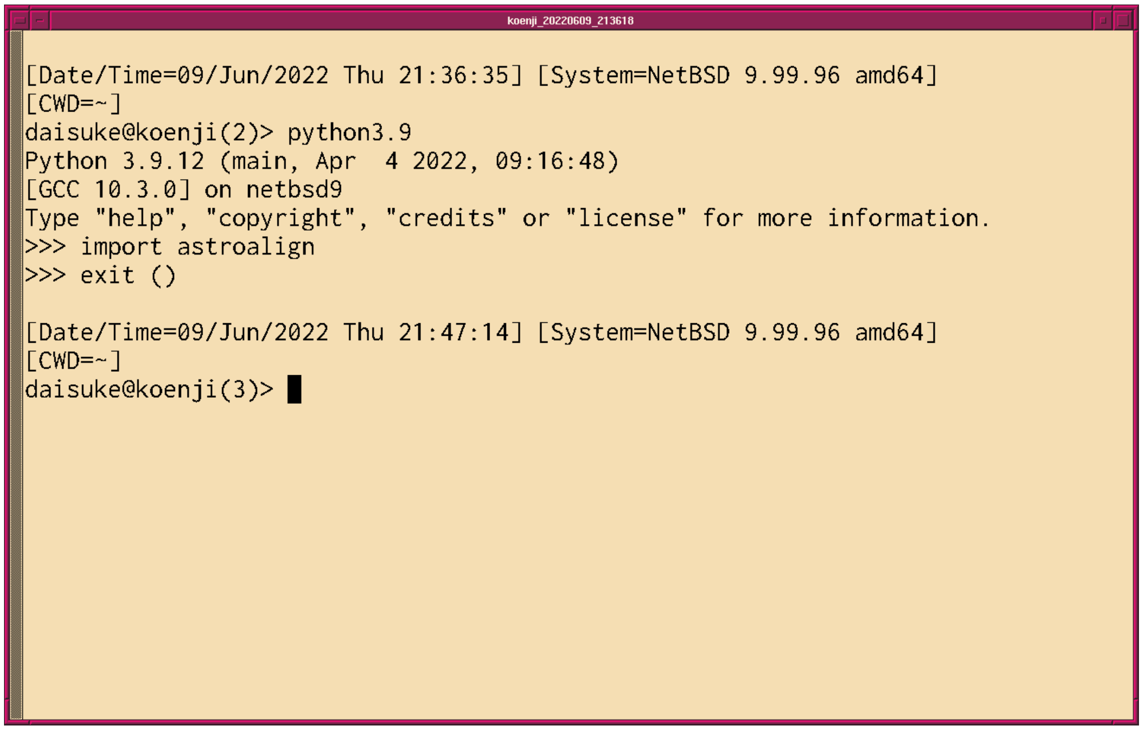
Figure 3: The official web page of astroalign package.

Try following to check whether or not you have **`astroalign`** installed on your computer. If the package is properly installed on your computer, then you can successfully import **`astroalign`**. (Fig. 4)

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

If you do not have **`astroalign`** installed on your computer, then you see a message like below.

```
% python3.9
Python 3.9.12 (main, Apr  6 2022, 23:48:29)
[GCC 10.3.0] on netbsd9
```



```
[Date/Time=09/Jun/2022 Thu 21:36:35] [System=NetBSD 9.99.96 amd64]
[CWD=~]
daisuke@koenji(2)> python3.9
Python 3.9.12 (main, Apr  4 2022, 09:16:48)
[GCC 10.3.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import astroalign
>>> exit ()

[Date/Time=09/Jun/2022 Thu 21:47:14] [System=NetBSD 9.99.96 amd64]
[CWD=~]
daisuke@koenji(3)> 
```

Figure 4: Importing astroalign package.

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import astroalign
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'astroalign'
>>> exit ()
```

3 Generating a set of synthetic images

We first try image alignment using synthetic images.

3.1 Generating (x, y) positions of stars

Make a Python script to generate (x, y) positions of artificial stars using `numpy.random` module.

Python Code 1: advobs202202_s17_01.py

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

#
# Time-stamp: <2022/06/09 21:54:16 (CST) daisuke>
#

# importing argparse module
import argparse

# importing pathlib module
import pathlib

# importing sys module
import sys
```

```
# importing numpy module
import numpy.random

# constructing parser object
desc   = 'generating random (x, y) positions of stars'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-n', '--number', type=int, default=10, \
                     help='number of stars (default: 10)')
parser.add_argument ('-x', '--size-x', type=int, default=1024, \
                     help='image size on x-axis (default: 2048)')
parser.add_argument ('-y', '--size-y', type=int, default=1024, \
                     help='image size on y-axis (default: 2048)')
parser.add_argument ('-a', '--flux-min', type=float, default=100.0, \
                     help='minimum flux of stars (default: 100)')
parser.add_argument ('-b', '--flux-max', type=float, default=100000.0, \
                     help='maximum flux of stars (default: 100000)')
parser.add_argument ('-o', '--file-output', default='', \
                     help='output file name')

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

# parameters
nstars      = args.number
size_x      = args.size_x
size_y      = args.size_y
flux_min    = args.flux_min
flux_max    = args.flux_max
file_output = args.file_output

# check of output file name
if (file_output == ''):
    # printing message
    print ("ERROR: Output file name must be specified.")
    # exit
    sys.exit ()

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

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

# generating random numbers
rng = numpy.random.default_rng ()
position_x = rng.uniform (0, size_x, nstars)
position_y = rng.uniform (0, size_y, nstars)
flux       = rng.uniform (flux_min, flux_max, nstars)

# writing data to file
with open (file_output, 'w') as fh_out:
    # for each object
    for i in range ( len (position_x) ):
        # writing x, y, flux to file
        fh_out.write ("%f %f %f\n" % (position_x[i], position_y[i], flux[i]) )
```

Run the script, and make a file.

```
% chmod a+x advobs202202_s17_01.py
% ./advobs202202_s17_01.py -h
usage: advobs202202_s17_01.py [-h] [-n NUMBER] [-x SIZE_X] [-y SIZE_Y]
                              [-a FLUX_MIN] [-b FLUX_MAX] [-o FILE_OUTPUT]

generating random (x, y) positions of stars

optional arguments:
  -h, --help            show this help message and exit
  -n NUMBER, --number NUMBER
                        number of stars (default: 10)
  -x SIZE_X, --size-x SIZE_X
                        image size on x-axis (default: 2048)
  -y SIZE_Y, --size-y SIZE_Y
                        image size on y-axis (default: 2048)
  -a FLUX_MIN, --flux-min FLUX_MIN
                        minimum flux of stars (default: 100)
  -b FLUX_MAX, --flux-max FLUX_MAX
                        maximum flux of stars (default: 100000)
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output file name

% ./advobs202202_s17_01.py -n 30 -x 2048 -y 2048 -o stars_0.list
% ls -lF
total 1
-rwxr-xr-x  1 daisuke  taiwan  2264 Jun  9 21:54 advobs202202_s17_01.py*
-rw-r--r--  1 daisuke  taiwan  1977 Jun  9 21:51 advobs202202_s17_01.py~
-rw-r--r--  1 daisuke  taiwan  1076 Jun  9 21:55 stars_0.list
% head stars_0.list
1616.809973 2033.648327 62793.320507
631.223798 186.639975 95412.168479
1546.672627 1774.023762 70287.958199
1148.151032 1002.596140 94800.110877
1564.186414 545.093320 34712.399189
333.561142 757.193734 19771.326261
161.159564 1839.545089 48499.255247
1067.458457 801.242040 14997.710172
1709.575160 1240.945627 13093.950116
420.503088 650.000404 88903.369385
```

3.2 Rotation and translation of coordinates

Make a Python script to give rotation and translation of (x, y) coordinates.

Python Code 2: advobs202202_s17_02.py

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

#
# Time-stamp: <2022/06/09 22:02:46 (CST) daisuke>
#
```

```
# importing argparse module
import argparse

# importing pathlib module
import pathlib

# importing sys module
import sys

# importing numpy module
import numpy

# constructing parser object
desc   = 'rotation and translation of (x, y) coordinates'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--file-input', default='', \
                     help='input file name')
parser.add_argument ('-o', '--file-output', default='', \
                     help='output file name')
parser.add_argument ('-c', '--centre-x', type=float, default=1024.0, \
                     help='x-coordinate of centre of rotation (default: 1024)')
parser.add_argument ('-d', '--centre-y', type=float, default=1024.0, \
                     help='y-coordinate of centre of rotation (default: 1024)')
parser.add_argument ('-r', '--rotate', type=float, default=45.0, \
                     help='rotation angle in degree (default: 45)')
parser.add_argument ('-s', '--shift-x', type=float, default=10.0, \
                     help='amount of shift on x-axis (default:10)')
parser.add_argument ('-t', '--shift-y', type=float, default=10.0, \
                     help='amount of shift on y-axis (default:10)')

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

# parameters
file_input  = args.file_input
file_output = args.file_output
centre_x    = args.centre_x
centre_y    = args.centre_y
rotate_deg  = args.rotate
shift_x     = args.shift_x
shift_y     = args.shift_y

# conversion from deg to rad
rotate_rad = numpy.deg2rad (rotate_deg)

# check of input file name
if (file_input == ''):
    # printing message
    print ("ERROR: Input file name must be specified.")
    # exit
    sys.exit ()

# check of output file name
if (file_output == ''):
    # printing message
    print ("ERROR: Output file name must be specified.")
    # exit
    sys.exit ()

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

# 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 ()

# list for new coordinates
list_new = []

# opening file for reading
with open (file_input, 'r') as fh_in:
    # reading file line-by-line
    for line in fh_in:
        # skipping line, if line starts with '#'
        if (line[0] == '#'):
            continue
        # splitting line
        (x_str, y_str, flux_str) = line.split ()
        # x, y, and flux
        x    = float (x_str)
        y    = float (y_str)
        flux = float (flux_str)
        # coordinate conversion (rotation and translation)
        x_0   = x - centre_x
        y_0   = y - centre_y
        x_1   = numpy.cos (rotate_rad) * x_0 - numpy.sin (rotate_rad) * y_0
        y_1   = numpy.sin (rotate_rad) * x_0 + numpy.cos (rotate_rad) * y_0
        x_2   = x_1 + shift_x
        y_2   = y_1 + shift_y
        x_new = x_2 + centre_x
        y_new = y_2 + centre_y
        # appending new positions to the list
        list_new.append ( (x_new, y_new, flux) )

# writing data to file
with open (file_output, 'w') as fh_out:
    # for each object
    for (x, y, flux) in list_new:
        # writing data
        fh_out.write ("%f %f %f\n" % (x, y, flux) )
```

Execute the script.

```
% chmod a+x advobs202202_s17_02.py
% ./advobs202202_s17_02.py -h
usage: advobs202202_s17_02.py [-h] [-i FILE_INPUT] [-o FILE_OUTPUT]
                              [-c CENTRE_X] [-d CENTRE_Y] [-r ROTATE]
                              [-s SHIFT_X] [-t SHIFT_Y]

rotation and translation of (x, y) coordinates

optional arguments:
  -h, --help            show this help message and exit
  -i FILE_INPUT, --file-input FILE_INPUT
                        input file name
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output file name
  -c CENTRE_X, --centre-x CENTRE_X
                        x-coordinate of centre of rotation (default: 1024)
  -d CENTRE_Y, --centre-y CENTRE_Y
                        y-coordinate of centre of rotation (default: 1024)
  -r ROTATE, --rotate ROTATE
                        rotation angle in degree (default: 45)
  -s SHIFT_X, --shift-x SHIFT_X
                        amount of shift on x-axis (default:10)
  -t SHIFT_Y, --shift-y SHIFT_Y
                        amount of shift on y-axis (default:10)

% ./advobs202202_s17_02.py -i stars_0.list -o stars_1.list -r 15 -s 55 -t 30
% ls -lF
total 1
-rwxr-xr-x  1 daisuke  taiwan  2264 Jun  9 21:54 advobs202202_s17_01.py*
-rw-r--r--  1 daisuke  taiwan  1977 Jun  9 21:51 advobs202202_s17_01.py~
-rwxr-xr-x  1 daisuke  taiwan  3582 Jun  9 22:02 advobs202202_s17_02.py*
-rw-r--r--  1 daisuke  taiwan  2988 Jun  9 21:57 advobs202202_s17_02.py~
-rw-r--r--  1 daisuke  taiwan  1076 Jun  9 21:55 stars_0.list
-rw-r--r--  1 daisuke  taiwan  1079 Jun  9 22:02 stars_1.list
% head stars_1.list
1390.294247 2182.675906 62793.320507
916.332045 143.514364 95412.168479
1389.742555 1913.744952 70287.958199
1204.460415 1065.458110 94800.110877
1724.730178 731.222201 34712.399189
481.141819 617.586211 19771.326261
34.481538 1618.436526 48499.255247
1178.631648 850.080210 14997.710172
1685.065093 1440.993292 13093.950116
592.864965 536.547637 88903.369385
```

3.3 Producing synthetic images

Make a Python script to read (x, y) coordinate file and produce a synthetic image.

Python Code 3: advobs202202_s17_03.py

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

#
# Time-stamp: <2022/06/09 22:07:44 (CST) daisuke>
#

# importing argparse module
import argparse
```

```
# importing sys module
import sys

# importing pathlib module
import pathlib

# importing numpy module
import numpy

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

# importing photutils module
import photutils.datasets

# constructing parser object
desc   = 'reading star list file and generating FITS image'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--file-input', default='', \
                     help='input file name')
parser.add_argument ('-o', '--file-output', default='', \
                     help='output file name')
parser.add_argument ('-f', '--fwhm', type=float, default=4.0, \
                     help='FWHM of PSF in pixel (default: 4)')
parser.add_argument ('-g', '--fwhm-stddev', type=float, default=0.1, \
                     help='stddev of FWHM of PSF 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 ('-x', '--size-x', type=int, default=2048, \
                     help='image size on x-axis (default: 2048)')
parser.add_argument ('-y', '--size-y', type=int, default=2048, \
                     help='image size on y-axis (default: 2048)')

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

# parameters
file_input  = args.file_input
file_output = args.file_output
fwhm        = args.fwhm
fwhm_stddev = args.fwhm_stddev
sky         = args.sky
sky_stddev  = args.sky_stddev
size_x      = args.size_x
size_y      = args.size_y

# image shape
image_shape = (size_x, size_y)

# check of input file name
if (file_input == ''):
    print ("Input file name must be specified.")
    sys.exit ()
```

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

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

# 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 ()

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

# making empty lists for data
list_x      = []
list_y      = []
list_flux   = []
list_psf_x = []
list_psf_y = []
list_theta = []

# opening file for reading
with open (file_input, 'r') as fh_in:
    # reading file line-by-line
    for line in fh_in:
        # skipping line, if line starts with '#'
        if (line[0] == '#'):
            continue
        # splitting line
        (x_str, y_str, flux_str) = line.split ()
        # x, y, and flux
        x    = float (x_str)
        y    = float (y_str)
        flux = float (flux_str)
        # random number generation for PSF
        rng       = numpy.random.default_rng ()
        psf_x     = rng.normal (loc=fwhm, scale=fwhm_stddev, size=1)
        psf_y     = rng.normal (loc=fwhm, scale=fwhm_stddev, size=1)
        theta_deg = rng.uniform (0.0, 360.0, 1)
        theta_rad = numpy.deg2rad (theta_deg)
        # adding data to lists
        list_flux.append (flux)
        list_x.append (x)
        list_y.append (y)
        list_psf_x.append (psf_x)
        list_psf_y.append (psf_y)
        list_theta.append (theta_rad)

# adding data to astropy table
table_stars['amplitude'] = list_flux
table_stars['x_mean']    = list_x
table_stars['y_mean']    = list_y
table_stars['x_stddev']  = list_psf_x
table_stars['y_stddev']  = list_psf_y
table_stars['theta']     = list_theta

# 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, \
                                                 stddev=sky_stddev)
# generating synthetic image
image = image_stars + 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)
```

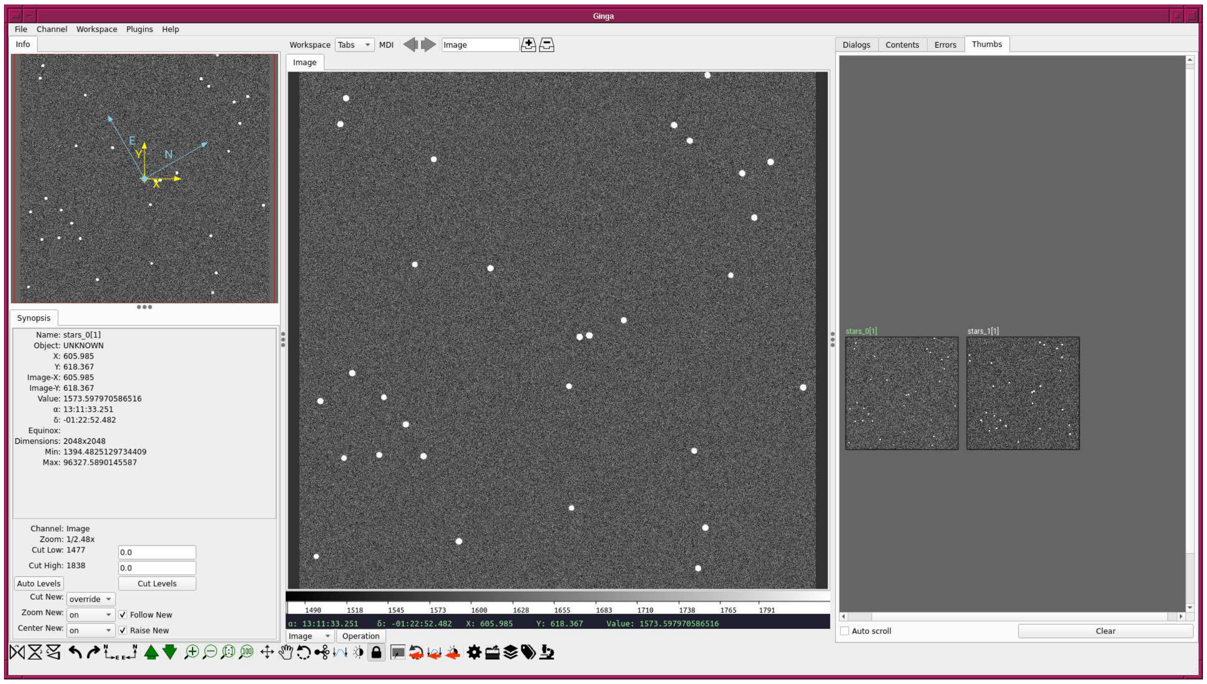
Run the script, and make FITS files.

```
% chmod a+x advobs202202_s17_03.py
% ./advobs202202_s17_03.py -h
usage: advobs202202_s17_03.py [-h] [-i FILE_INPUT] [-o FILE_OUTPUT] [-f FWHM]
                              [-g FWHM_STDDEV] [-s SKY] [-e SKY_STDDEV]
                              [-x SIZE_X] [-y SIZE_Y]

reading star list file and generating FITS image

optional arguments:
  -h, --help            show this help message and exit
  -i FILE_INPUT, --file-input FILE_INPUT
                        input file name
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output file name
  -f FWHM, --fwhm FWHM  FWHM of PSF in pixel (default: 4)
  -g FWHM_STDDEV, --fwhm-stddev FWHM_STDDEV
                        stddev of FWHM of PSF 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)
  -x SIZE_X, --size-x SIZE_X
                        image size on x-axis (default: 2048)
  -y SIZE_Y, --size-y SIZE_Y
                        image size on y-axis (default: 2048)

% ./advobs202202_s17_03.py -i stars_0.list -o stars_0.fits -f 3.5 -s 1600 -e 40
% ./advobs202202_s17_03.py -i stars_1.list -o stars_1.fits -f 4.5 -s 2500 -e 50
% ls -lF
total 65
-rwxr-xr-x  1 daisuke  taiwan      2264 Jun  9 21:54 advobs202202_s17_01.py*
-rw-r--r--  1 daisuke  taiwan      1977 Jun  9 21:51 advobs202202_s17_01.py~
-rwxr-xr-x  1 daisuke  taiwan      3582 Jun  9 22:02 advobs202202_s17_02.py*
-rw-r--r--  1 daisuke  taiwan      2988 Jun  9 21:57 advobs202202_s17_02.py~
-rwxr-xr-x  1 daisuke  taiwan      4733 Jun  9 22:07 advobs202202_s17_03.py*
-rw-r--r--  1 daisuke  taiwan      4451 Jun  9 22:04 advobs202202_s17_03.py~
-rw-r--r--  1 daisuke  taiwan  33560640 Jun  9 22:08 stars_0.fits
-rw-r--r--  1 daisuke  taiwan      1076 Jun  9 21:55 stars_0.list
-rw-r--r--  1 daisuke  taiwan  33560640 Jun  9 22:09 stars_1.fits
-rw-r--r--  1 daisuke  taiwan      1079 Jun  9 22:02 stars_1.list
% file *.fits
stars_0.fits: FITS image data, 8-bit, character or unsigned binary integer
stars_1.fits: FITS image data, 8-bit, character or unsigned binary integer
```

3.4 Visual inspection of FITS files

Use Ginga to check the FITS files. (Fig. 5 and 6)

```
% ginga stars_?.fits
```



Name: stars_0[1]
Object: UNKNOWN
X: 605.985
Y: 618.367
Image-X: 605.985
Image-Y: 618.367
Value: 1573.597970586516
α: 13:11:33.251
δ: -01:22:52.482
Equinox:
Dimensions: 2048x2048
Min: 1394.4825129734409
Max: 96327.5890145587

Channel: Image
Zoom: 1/2.48x
Cut Low: 1477 0.0
Cut High: 1838 0.0
Cut New: override
Zoom New: on ✓ Follow New
Center New: on ✓ Raise New

α: 13:11:33.251 δ: -01:22:52.482 X: 605.985 Y: 618.367 Value: 1573.597970586516

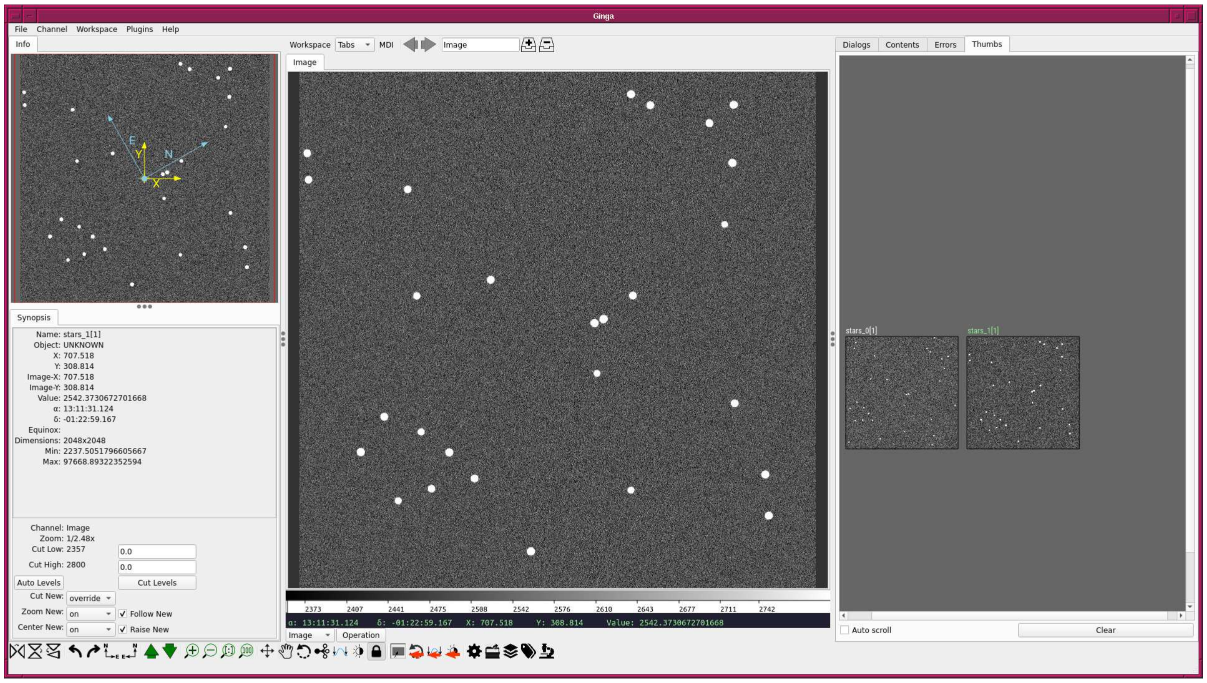
Figure 5: First synthetic image.

4 Source extraction

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

Python Code 4: advobs202202_s17_04.py

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



```
Name: stars_1[1]
Object: UNKNOWN
X: 707.518
Y: 308.814
Image-X: 707.518
Image-Y: 308.814
Value: 2542.3730672701668
α: 13:11:31.124
δ: -01:22:59.167
Equinox:
Dimensions: 2048x2048
Min: 2237.5051796605667
Max: 97668.89322352594

Channel: Image
Zoom: 1/2.48x
Cut Low: 2357     0.0
Cut High: 2800    0.0
Auto Levels   Cut Levels
Cut New: override
Zoom New: on     ✓ Follow New
Center New: on   ✓ Raise New

α: 13:11:31.124   δ: -01:22:59.167   X: 707.518   Y: 308.814   Value: 2542.3730672701668
```

Figure 6: Second synthetic image.

```
#
# Time-stamp: <2022/06/09 22:27:06 (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 and deblending'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--file-input', default='', \
                     help='input FITS file name')
parser.add_argument ('-o', '--file-catalogue', default='', \
                     help='output catalogue file name')
parser.add_argument ('-g', '--file-fig', default='', \
                     help='output figure 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 ('-c', '--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)')
parser.add_argument ('-r', '--radius', type=float, default=30.0, \
                     help='radius of aperture in pixel (default: 30)')

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

# file names
file_input     = args.file_input
file_catalogue = args.file_catalogue
file_fig       = args.file_fig

# 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
radius            = args.radius

# making pathlib objects
path_input     = pathlib.Path (file_input)
path_catalogue = pathlib.Path (file_catalogue)
path_fig       = pathlib.Path (file_fig)

# check of input file name
if not (path_input.suffix == '.fits'):
    # printing message
    print ("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 figure file name
if not ( (path_fig.suffix == '.eps') or (path_fig.suffix == '.pdf') \
         or (path_fig.suffix == '.png') or (path_fig.suffix == '.ps') ):
    # printing message
    print ("Figure file name 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 (path_catalogue.exists ()):
    # printing message
    print ("ERROR: Catalogue file '%s' exists." % (file_catalogue) )
    # exit
    sys.exit ()

# existence check of fig file
if (path_fig.exists ()):
    # printing message
    print ("ERROR: Fig file '%s' exists." % (file_fig) )
    # 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')

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

# 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_fig, dpi=225)
```

Execute the script, and generate a list of sources detected from the image.

```
% chmod a+x advobs202202_s17_04.py
% ./advobs202202_s17_04.py -h
usage: advobs202202_s17_04.py [-h] [-i FILE_INPUT] [-o FILE_CATALOGUE]
                              [-g FILE_FIG] [-t THRESHOLD]
                              [-u THRESHOLD_FOR_SKY] [-n NPIXELS]
                              [-s DILATE_SIZE] [-m MAXITERS]
                              [-c SIGMA_CLIPPING] [-k GAUSSIAN_FWHM]
                              [-a KERNEL_SIZE] [-r RADIUS]

source extraction using image segmentation and deblending

optional arguments:
  -h, --help            show this help message and exit
  -i FILE_INPUT, --file-input FILE_INPUT
                        input FITS file name
  -o FILE_CATALOGUE, --file-catalogue FILE_CATALOGUE
                        output catalogue file name
  -g FILE_FIG, --file-fig FILE_FIG
                        output figure 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)
  -c 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)
  -r RADIUS, --radius RADIUS
                        radius of aperture in pixel (default: 30)

% ./advobs202202_s17_04.py -i stars_0.fits -o stars_0.cat -g stars_0.pdf
% ./advobs202202_s17_04.py -i stars_1.fits -o stars_1.cat -g stars_1.pdf
% ls -lF stars_?.*
```

```
-rw-r--r--  1 daisuke  taiwan      7084 Jun  9 22:27 stars_0.cat
-rw-r--r--  1 daisuke  taiwan  33560640 Jun  9 22:08 stars_0.fits
-rw-r--r--  1 daisuke  taiwan      1076 Jun  9 21:55 stars_0.list
-rw-r--r--  1 daisuke  taiwan    616251 Jun  9 22:27 stars_0.pdf
-rw-r--r--  1 daisuke  taiwan      6399 Jun  9 22:27 stars_1.cat
-rw-r--r--  1 daisuke  taiwan  33560640 Jun  9 22:09 stars_1.fits
-rw-r--r--  1 daisuke  taiwan      1079 Jun  9 22:02 stars_1.list
-rw-r--r--  1 daisuke  taiwan    615190 Jun  9 22:27 stars_1.pdf
% head -3 stars_?.cat
==> stars_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 1579.5158663773323 79.60825331486821 None 1568 1591 68 91 463.0 3.811472054367
469 3.778388159213444 74.7922458462029 0.13147175693555485 1647.5138648562183 70
424.78787722586 0.0 5610724.022421351 nan 6901986.765104825 nan
2 65.1484201133629 126.5362043353147 None 55 75 117 137 338.0 4.366354210004487
4.192581007979327 26.39474544770332 0.27930731348288224 1658.1646300517386 10527
.360131005214 0.0 1214200.064619423 nan 7683957.506047544 nan

==> stars_1.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
1 916.3336586394673 143.50946518895145 None 900 933 127 160 874.0 5.379606926460
29 5.212604149878997 15.851596692326284 0.2472319618384994 2569.901381435816 971
46.60670113146 0.0 14772379.020508548 nan 19316008.305820376 nan
2 1860.0076379272853 285.5751062291951 None 1844 1876 270 301 822.0 5.3873459932
74005 5.337240234808144 20.14349946587097 0.13606900990836254 2501.392021569849
72257.37168514723 0.0 11156576.619202297 nan 17582975.991104905 nan
```

Display PDF files, and check the results of source extraction. (Fig. 7 and 8)

```
% okular stars_0.pdf
% okular stars_1.pdf
```

5 Finding star-to-star correspondence

Make a Python script to find star-to-star correspondence of two images.

Python Code 5: advobs202202_s17_05.py

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

#
# Time-stamp: <2022/06/09 22:34:25 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib
```

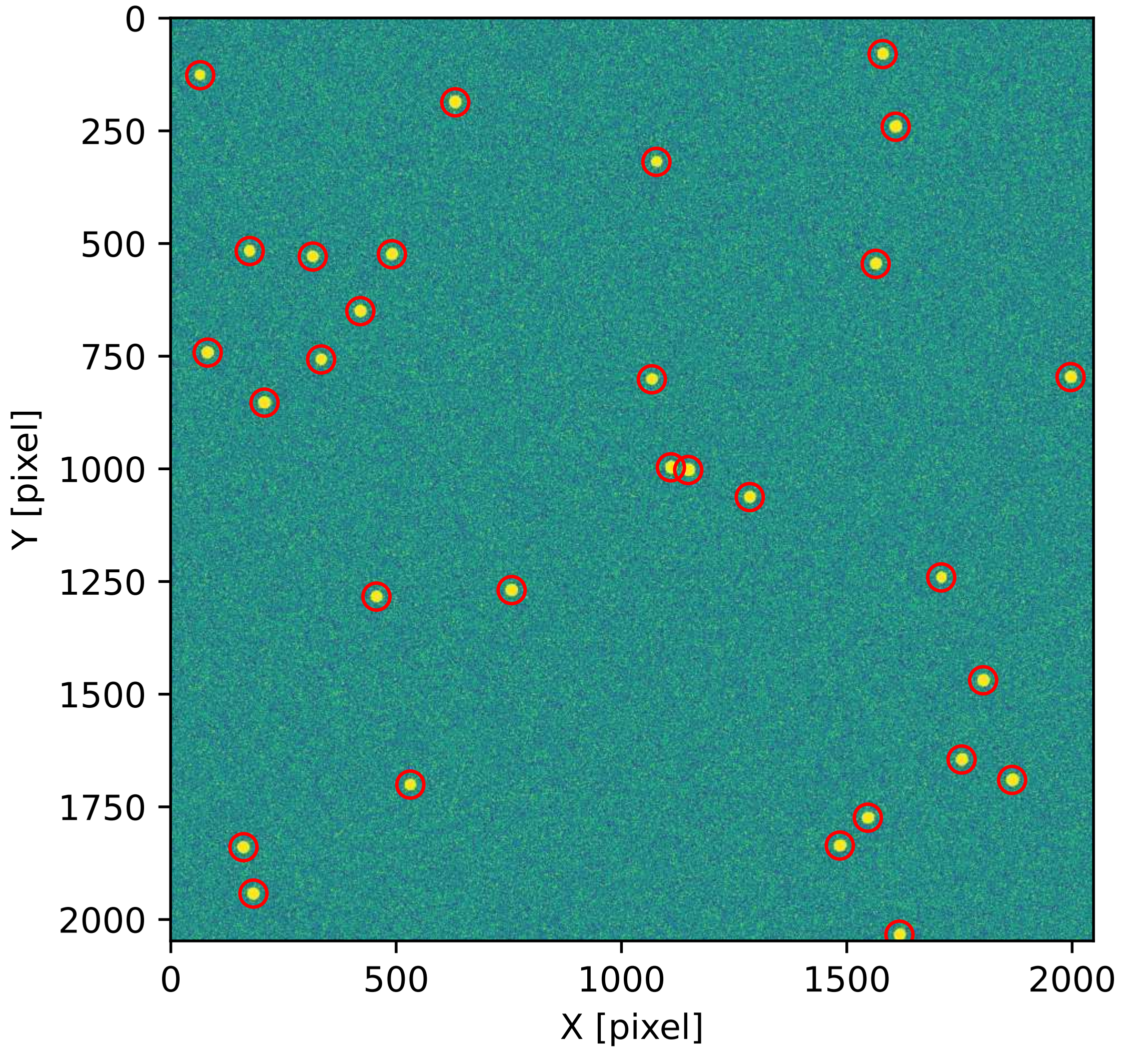


Figure 7: Sources detected from the FITS file `stars_0.fits`.

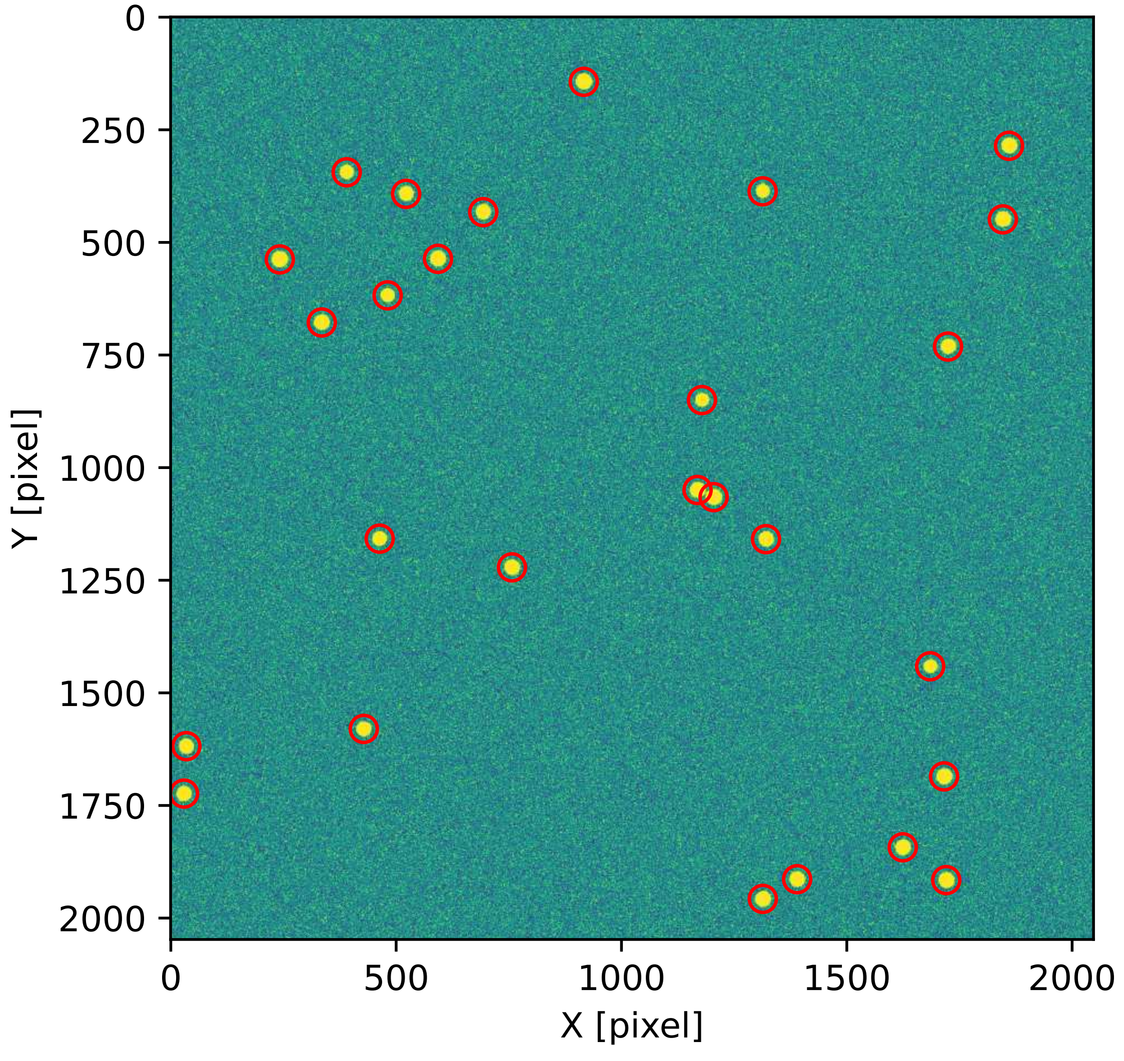


Figure 8: Sources detected from the FITS file `stars_1.fits`.

```
# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.table

# importing astroalign module
import astroalign

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

# constructing parser object
desc   = 'finding star-to-star correspondence'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('catalogue1', nargs=1, help='catalogue file 1')
parser.add_argument ('catalogue2', nargs=1, help='catalogue file 2')

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

# file names
file_cat1 = args.catalogue1[0]
file_cat2 = args.catalogue2[0]

# making pathlib objects
path_cat1 = pathlib.Path (file_cat1)
path_cat2 = pathlib.Path (file_cat2)

# check of catalogue file name
if not ( (path_cat1.suffix == '.cat') and (path_cat2.suffix == '.cat') ):
    # printing message
    print ("Input file must be a catalogue file (*.cat).")
    print ("catalogue file 1 = %s" % file_cat1)
    print ("catalogue file 2 = %s" % file_cat2)
    # exit
    sys.exit ()

# existence check of cat1 file
if not (path_cat1.exists ()):
    # printing message
    print ("ERROR: catalogue file 1 '%s' does not exist." % (file_cat1) )
    # exit
    sys.exit ()

# existence check of cat2 file
if not (path_cat2.exists ()):
    # printing message
    print ("ERROR: catalogue file 2 '%s' does not exist." % (file_cat2) )
    # exit
    sys.exit ()

# reading catalogue from a file
```

```
table_source1 = astropy.table.Table.read (file_cat1, \
                                          format='ascii.commented_header')
table_source2 = astropy.table.Table.read (file_cat2, \
                                          format='ascii.commented_header')

# (x, y) coordinates of sources
list_source1_x = list (table_source1['xcentroid'])
list_source1_y = list (table_source1['ycentroid'])
list_source2_x = list (table_source2['xcentroid'])
list_source2_y = list (table_source2['ycentroid'])
position_1     = numpy.transpose ( (list_source1_x, list_source1_y) )
position_2     = numpy.transpose ( (list_source2_x, list_source2_y) )

# finding star-to-star matching
transf, (list_matched_2, list_matched_1) \
    = astroalign.find_transform (position_2, position_1)

# transformation
list_matched_2_aligned \
    = astroalign.matrix_transform (list_matched_2, transf.params)

# printing results
print ("#")
print ("# result of image alignment")
print ("#")
print ("#   date/time = %s" % now)
print ("#")
print ("# input files")
print ("#")
print ("#   catalogue file 1 = %s" % file_cat1)
print ("#   catalogue file 2 = %s" % file_cat2)
print ("#")
print ("# transformation matrix")
print ("#")
print ("# [")
print ("#   [%f, %f, %f]," \
       % (transf.params[0][0], transf.params[0][1], transf.params[0][2]) )
print ("#   [%f, %f, %f]," \
       % (transf.params[1][0], transf.params[1][1], transf.params[1][2]) )
print ("#   [%f, %f, %f]" \
       % (transf.params[2][0], transf.params[2][1], transf.params[2][2]) )
print ("# ]")
print ("#")
print ("#")
print ("# list of matched stars")
print ("#")
for i in range ( len (list_matched_1) ):
    print ("(%10.4f, %10.4f) on 1st image ==> (%10.4f, %10.4f) on 2nd image" \
           % (list_matched_1[i][0], list_matched_1[i][1], \
              list_matched_2[i][0], list_matched_2[i][1]) )
```

Run the script, and carry out matching.

```
% chmod a+x advobs202202_s17_05.py
% ./advobs202202_s17_05.py -h
usage: advobs202202_s17_05.py [-h] catalogue1 catalogue2

finding star-to-star correspondence
```

```
positional arguments:
  catalogue1  catalogue file 1
  catalogue2  catalogue file 2

optional arguments:
  -h, --help  show this help message and exit

% ./advobs202202_s17_05.py stars_0.cat stars_1.cat
#
# result of image alignment
#
#   date/time = 2022-06-09 22:37:15.105031
#
# input files
#
#   catalogue file 1 = stars_0.cat
#   catalogue file 2 = stars_1.cat
#
# transformation matrix
#
# [
#   [0.965930, 0.258819, -291.031611],
#   [-0.258819, 0.965930, 285.168282],
#   [0.000000, 0.000000, 1.000000]
# ]
#
#
# list of matched stars
#
( 1866.8354,  1690.2168) on 1st image ==> ( 1720.6830,  1915.6673) on 2nd image
( 1077.5368,   318.7460) on 1st image ==> ( 1313.2339,   386.6019) on 2nd image
( 1067.4470,   801.2203) on 1st image ==> ( 1178.6199,   850.0304) on 2nd image
( 1148.1532,  1002.5993) on 1st image ==> ( 1204.4642,  1065.4658) on 2nd image
( 1754.8088,  1644.8986) on 1st image ==> ( 1624.1923,  1842.8884) on 2nd image
(  631.2147,   186.6385) on 1st image ==> (  916.3337,   143.5095) on 2nd image
( 1579.5159,    79.6083) on 1st image ==> ( 1860.0076,   285.5751) on 2nd image
(  333.5968,   757.1984) on 1st image ==> (  481.1349,   617.5982) on 2nd image
( 1564.1786,   545.0747) on 1st image ==> ( 1724.6907,   731.2205) on 2nd image
(  175.0280,   516.6916) on 1st image ==> (  390.2882,   344.3251) on 2nd image
(  531.4456,  1700.6104) on 1st image ==> (  428.1424,  1580.0668) on 2nd image
(  456.2789,  1283.4888) on 1st image ==> (  463.4521,  1157.7099) on 2nd image
( 1484.5031,  1835.8843) on 1st image ==> ( 1313.6694,  1957.4054) on 2nd image
( 1802.5422,  1469.1342) on 1st image ==> ( 1715.7960,  1685.4752) on 2nd image
(  490.6636,   523.7812) on 1st image ==> (  693.2843,   432.8002) on 2nd image
(  183.3418,  1942.4515) on 1st image ==> (   29.2805,  1723.6011) on 2nd image
(  207.9840,   853.1035) on 1st image ==> (  335.0328,   677.7534) on 2nd image
(  420.5189,   650.0014) on 1st image ==> (  592.8722,   536.5421) on 2nd image
( 1546.6757,  1774.0108) on 1st image ==> ( 1389.7549,  1913.7522) on 2nd image
( 1709.5695,  1240.9216) on 1st image ==> ( 1685.0338,  1441.0519) on 2nd image
(  314.8656,   528.8662) on 1st image ==> (  522.1874,   392.2157) on 2nd image
(  161.1666,  1839.5355) on 1st image ==> (   34.5014,  1618.4186) on 2nd image
(   81.7757,   742.0294) on 1st image ==> (  241.8532,   537.7799) on 2nd image
(  756.3128,  1269.0433) on 1st image ==> (  757.0154,  1221.4139) on 2nd image
( 1109.6329,   996.5852) on 1st image ==> ( 1168.8099,  1049.6668) on 2nd image
( 1608.5440,   240.8841) on 1st image ==> ( 1846.2962,   448.8502) on 2nd image
( 1284.6825,  1062.6569) on 1st image ==> ( 1320.8116,  1158.8328) on 2nd image
```

6 Check of star-to-star correspondence

Make a Python script to check star-to-star correspondence.

Python Code 6: advobs202202_s17_06.py

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

#
# Time-stamp: <2022/06/09 22:44:50 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing datetime module
import datetime

# importing numpy module
import numpy

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

# importing astroalign module
import astroalign

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

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

# constructing parser object
desc   = 'checking results of star-to-star correspondence'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-o', '--file-output', default='', \
                     help='output figure file')
parser.add_argument ('catalogue1', nargs=1, help='catalogue file 1')
parser.add_argument ('catalogue2', nargs=1, help='catalogue file 2')
parser.add_argument ('fits1', nargs=1, help='FITS file 1')
parser.add_argument ('fits2', nargs=1, help='FITS file 2')

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

# file names
file_cat1  = args.catalogue1[0]
file_cat2  = args.catalogue2[0]
file_fits1 = args.fits1[0]
```

```
file_fits2 = args.fits2[0]
file_fig   = args.file_output

# making pathlib objects
path_cat1  = pathlib.Path (file_cat1)
path_cat2  = pathlib.Path (file_cat2)
path_fits1 = pathlib.Path (file_fits1)
path_fits2 = pathlib.Path (file_fits2)
path_fig   = pathlib.Path (file_fig)

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

# check of catalogue file name
if not ( (path_cat1.suffix == '.cat') and (path_cat2.suffix == '.cat') ):
    # printing message
    print ("Input file must be a catalogue file (*.cat).")
    print ("catalogue file 1 = %s" % file_cat1)
    print ("catalogue file 2 = %s" % file_cat2)
    # exit
    sys.exit ()

# check of FITS file name
if not ( (path_fits1.suffix == '.fits') and (path_fits2.suffix == '.fits') ):
    # printing message
    print ("Input file must be a FITS file (*.fits).")
    print ("FITS file 1 = %s" % file_fits1)
    print ("FITS file 2 = %s" % file_fits2)
    # exit
    sys.exit ()

# existence checks
if not (path_cat1.exists ()):
    # printing message
    print ("ERROR: file '%s' does not exist." % file_cat1)
    # exit
    sys.exit ()
if not (path_cat2.exists ()):
    # printing message
    print ("ERROR: file '%s' does not exist." % file_cat2)
    # exit
    sys.exit ()
if not (path_fits1.exists ()):
    # printing message
    print ("ERROR: file '%s' does not exist." % file_fits1)
    # exit
    sys.exit ()
if not (path_fits2.exists ()):
    # printing message
    print ("ERROR: file '%s' does not exist." % file_fits2)
    # exit
    sys.exit ()
if (path_fig.exists ()):
    # printing message
    print ("ERROR: file '%s' exists." % file_fig)
    # exit
    sys.exit ()

# function to read a FITS file
def read_fits (file_fits):
    # reading FITS file
    with astropy.io.fits.open (file_fits) 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
    # returning header and image
    return (header, image)

# reading catalogue from a file
table_source1 = astropy.table.Table.read (file_cat1, \
                                          format='ascii.commented_header')
table_source2 = astropy.table.Table.read (file_cat2, \
                                          format='ascii.commented_header')

# (x, y) coordinates of sources
list_source1_x = list (table_source1['xcentroid'])
list_source1_y = list (table_source1['ycentroid'])
list_source2_x = list (table_source2['xcentroid'])
list_source2_y = list (table_source2['ycentroid'])
position_1 = numpy.transpose ( (list_source1_x, list_source1_y) )
position_2 = numpy.transpose ( (list_source2_x, list_source2_y) )

# finding star-to-star matching
transf, (list_matched_2, list_matched_1) \
    = astroalign.find_transform (position_2, position_1)

# transformation
list_matched_2_aligned \
    = astroalign.matrix_transform (list_matched_2, transf.params)

# printing results
print ("#")
print ("# result of image alignment")
print ("#")
print ("#   date/time = %s" % now)
print ("#")
print ("# input files")
print ("#")
print ("#   catalogue file 1 = %s" % file_cat1)
print ("#   catalogue file 2 = %s" % file_cat2)
print ("#")
print ("# transformation matrix")
print ("#")
print ("# [")
print ("#   [%f, %f, %f]," \
       % (transf.params[0][0], transf.params[0][1], transf.params[0][2]) )
print ("#   [%f, %f, %f]," \
       % (transf.params[1][0], transf.params[1][1], transf.params[1][2]) )
print ("#   [%f, %f, %f]" \
       % (transf.params[2][0], transf.params[2][1], transf.params[2][2]) )
print ("# ]")
print ("#")
print ("#")
print ("# list of matched stars")
print ("#")
for i in range ( len (list_matched_1) ):
    print ("(%10.4f, %10.4f) on 1st image ==> (%10.4f, %10.4f) on 2nd image" \
           % (list_matched_1[i][0], list_matched_1[i][1], \
              list_matched_2[i][0], list_matched_2[i][1]) )

# reading FITS files
(header1, image1) = read_fits (file_fits1)
(header2, image2) = read_fits (file_fits2)

# marker and colour names for matplotlib
markers = ['o', 'v', '^', 's', 'p', 'h', '8']
colours = ['maroon', 'red', 'coral', 'bisque', 'orange', \
           'wheat', 'yellow', 'green', 'lime', 'aqua', \
           'skyblue', 'blue', 'indigo', 'violet', 'pink']

# 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 first image
norm1 \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image1) )
im1 = ax1.imshow (image1, origin='lower', cmap='bone', norm=norm1)
for i in range ( len (list_matched_1) ):
    i_marker = i % len (markers)
    i_colour = i % len (colours)
    ax1.plot (list_matched_1[i][0], list_matched_1[i][1], \
              marker=markers[i_marker], color=colours[i_colour], \
              markersize=8, fillstyle='none')
ax1.set_title ('First Image')

# plotting second image
norm2 \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image2) )
im2 = ax2.imshow (image2, origin='lower', cmap='bone', norm=norm2)
for i in range ( len (list_matched_2) ):
    i_marker = i % len (markers)
    i_colour = i % len (colours)
    ax2.plot (list_matched_2[i][0], list_matched_2[i][1], \
              marker=markers[i_marker], color=colours[i_colour], \
              markersize=8, fillstyle='none')
ax2.set_title ('Second Image')

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

Execute the script.

```
% chmod a+x advobs202202_s17_06.py
% ./advobs202202_s17_06.py -h
usage: advobs202202_s17_06.py [-h] [-o FILE_OUTPUT]
                              catalogue1 catalogue2 fits1 fits2

checking results of star-to-star correspondence

positional arguments:
  catalogue1            catalogue file 1
  catalogue2            catalogue file 2
  fits1                 FITS file 1
  fits2                 FITS file 2

optional arguments:
  -h, --help            show this help message and exit
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output figure file

% ./advobs202202_s17_06.py -o match_0.pdf stars_0.cat stars_1.cat \
? stars_0.fits stars_1.fits
#
# result of image alignment
#
#   date/time = 2022-06-09 22:46:08.882954
#
# input files
#
#   catalogue file 1 = stars_0.cat
#   catalogue file 2 = stars_1.cat
#
# transformation matrix
#
# [
#   [0.965930, 0.258819, -291.031611],
#   [-0.258819, 0.965930, 285.168282],
#   [0.000000, 0.000000, 1.000000]
# ]
#
#
# list of matched stars
#
( 1866.8354,  1690.2168) on 1st image ==> ( 1720.6830,  1915.6673) on 2nd image
( 1077.5368,   318.7460) on 1st image ==> ( 1313.2339,   386.6019) on 2nd image
( 1067.4470,   801.2203) on 1st image ==> ( 1178.6199,   850.0304) on 2nd image
( 1148.1532,  1002.5993) on 1st image ==> ( 1204.4642,  1065.4658) on 2nd image
( 1754.8088,  1644.8986) on 1st image ==> ( 1624.1923,  1842.8884) on 2nd image
(  631.2147,   186.6385) on 1st image ==> (  916.3337,   143.5095) on 2nd image
( 1579.5159,    79.6083) on 1st image ==> ( 1860.0076,   285.5751) on 2nd image
(  333.5968,   757.1984) on 1st image ==> (  481.1349,   617.5982) on 2nd image
( 1564.1786,   545.0747) on 1st image ==> ( 1724.6907,   731.2205) on 2nd image
(  175.0280,   516.6916) on 1st image ==> (  390.2882,   344.3251) on 2nd image
(  531.4456,  1700.6104) on 1st image ==> (  428.1424,  1580.0668) on 2nd image
(  456.2789,  1283.4888) on 1st image ==> (  463.4521,  1157.7099) on 2nd image
( 1484.5031,  1835.8843) on 1st image ==> ( 1313.6694,  1957.4054) on 2nd image
( 1802.5422,  1469.1342) on 1st image ==> ( 1715.7960,  1685.4752) on 2nd image
(  490.6636,   523.7812) on 1st image ==> (  693.2843,   432.8002) on 2nd image
(  183.3418,  1942.4515) on 1st image ==> (   29.2805,  1723.6011) on 2nd image
(  207.9840,   853.1035) on 1st image ==> (  335.0328,   677.7534) on 2nd image
(  420.5189,   650.0014) on 1st image ==> (  592.8722,   536.5421) on 2nd image
( 1546.6757,  1774.0108) on 1st image ==> ( 1389.7549,  1913.7522) on 2nd image
( 1709.5695,  1240.9216) on 1st image ==> ( 1685.0338,  1441.0519) on 2nd image
(  314.8656,   528.8662) on 1st image ==> (  522.1874,   392.2157) on 2nd image
(  161.1666,  1839.5355) on 1st image ==> (   34.5014,  1618.4186) on 2nd image
(   81.7757,   742.0294) on 1st image ==> (  241.8532,   537.7799) on 2nd image
(  756.3128,  1269.0433) on 1st image ==> (  757.0154,  1221.4139) on 2nd image
( 1109.6329,   996.5852) on 1st image ==> ( 1168.8099,  1049.6668) on 2nd image
( 1608.5440,   240.8841) on 1st image ==> ( 1846.2962,   448.8502) on 2nd image
( 1284.6825,  1062.6569) on 1st image ==> ( 1320.8116,  1158.8328) on 2nd image
% ls -lF match_0.pdf
-rw-r--r--  1 daisuke  taiwan  427388 Jun  9 22:46 match_0.pdf
```

Check the PDF file. (Fig. 9)

```
% okular match_0.pdf
```

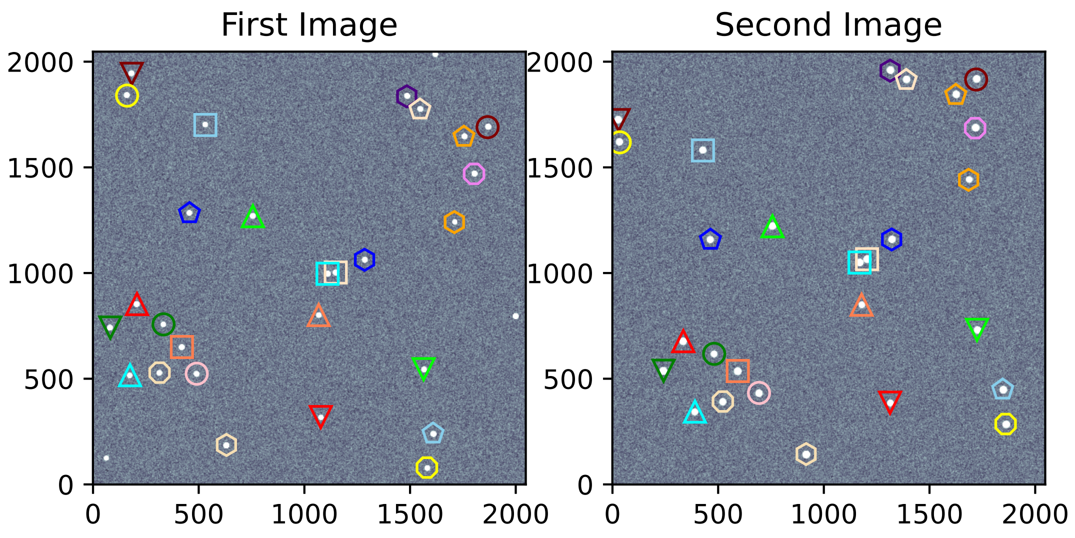


Figure 9: Result of finding star-to-star correspondence.

7 Aligining the image

Make a Python script to align an image to the reference image.

Python Code 7: advobs202202_s17_07.py

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

#
# Time-stamp: <2022/06/09 22:52:51 (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

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

# importing scikit-image module
import skimage.transform

# importing astroalign module
import astroalign

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

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

# constructing parser object
desc   = 'aligning image'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-o', '--file-output', default='', \
                     help='output figure file')
parser.add_argument ('catalogue1', nargs=1, help='catalogue file 1')
parser.add_argument ('catalogue2', nargs=1, help='catalogue file 2')
parser.add_argument ('fits1', nargs=1, help='FITS file 1')
parser.add_argument ('fits2', nargs=1, help='FITS file 2')

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

# file names
file_cat1  = args.catalogue1[0]
file_cat2  = args.catalogue2[0]
file_fits1 = args.fits1[0]
file_fits2 = args.fits2[0]
file_fig   = args.file_output
```

```
# making pathlib objects
path_cat1  = pathlib.Path (file_cat1)
path_cat2  = pathlib.Path (file_cat2)
path_fits1 = pathlib.Path (file_fits1)
path_fits2 = pathlib.Path (file_fits2)
path_fig   = pathlib.Path (file_fig)

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

# check of catalogue file name
if not ( (path_cat1.suffix == '.cat') and (path_cat2.suffix == '.cat') ):
    # printing message
    print ("Input file must be a catalogue file (*.cat).")
    print ("catalogue file 1 = %s" % file_cat1)
    print ("catalogue file 2 = %s" % file_cat2)
    # exit
    sys.exit ()

# check of FITS file name
if not ( (path_fits1.suffix == '.fits') and (path_fits2.suffix == '.fits') ):
    # printing message
    print ("Input file must be a FITS file (*.fits).")
    print ("FITS file 1 = %s" % file_fits1)
    print ("FITS file 2 = %s" % file_fits2)
    # exit
    sys.exit ()

# existence checks
if not (path_cat1.exists ()):
    # printing message
    print ("ERROR: file '%s' does not exist." % file_cat1)
    # exit
    sys.exit ()
if not (path_cat2.exists ()):
    # printing message
    print ("ERROR: file '%s' does not exist." % file_cat2)
    # exit
    sys.exit ()
if not (path_fits1.exists ()):
    # printing message
    print ("ERROR: file '%s' does not exist." % file_fits1)
    # exit
    sys.exit ()
if not (path_fits2.exists ()):
    # printing message
    print ("ERROR: file '%s' does not exist." % file_fits2)
    # exit
    sys.exit ()
if (path_fig.exists ()):
    # printing message
    print ("ERROR: file '%s' exists." % file_fig)
    # exit
    sys.exit ()

# function to read a FITS file
def read_fits (file_fits):
    # reading FITS file
    with astropy.io.fits.open (file_fits) 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
    # returning header and image
    return (header, image)

# reading catalogue from a file
table_source1 = astropy.table.Table.read (file_cat1, \
                                          format='ascii.commented_header')
table_source2 = astropy.table.Table.read (file_cat2, \
                                          format='ascii.commented_header')

# (x, y) coordinates of sources
list_source1_x = list (table_source1['xcentroid'])
list_source1_y = list (table_source1['ycentroid'])
list_source2_x = list (table_source2['xcentroid'])
list_source2_y = list (table_source2['ycentroid'])
position_1 = numpy.transpose ( (list_source1_x, list_source1_y) )
position_2 = numpy.transpose ( (list_source2_x, list_source2_y) )

# finding star-to-star matching
transf, (list_matched_2, list_matched_1) \
    = astroalign.find_transform (position_2, position_1)

# transformation
list_matched_2_aligned \
    = astroalign.matrix_transform (list_matched_2, transf.params)

# printing results
print ("#")
print ("# result of image alignment")
print ("#")
print ("#   date/time = %s" % now)
print ("#")
print ("# input files")
print ("#")
print ("#   catalogue file 1 = %s" % file_cat1)
print ("#   catalogue file 2 = %s" % file_cat2)
print ("#")
print ("# transformation matrix")
print ("#")
print ("# [")
print ("#   [%f, %f, %f]," \
       % (transf.params[0][0], transf.params[0][1], transf.params[0][2]) )
print ("#   [%f, %f, %f]," \
       % (transf.params[1][0], transf.params[1][1], transf.params[1][2]) )
print ("#   [%f, %f, %f]" \
       % (transf.params[2][0], transf.params[2][1], transf.params[2][2]) )
print ("# ]")
print ("#")
print ("#")
print ("# list of matched stars")
print ("#")
for i in range ( len (list_matched_1) ):
    print ("(%10.4f, %10.4f) on 1st image ==> (%10.4f, %10.4f) on 2nd image" \
           % (list_matched_1[i][0], list_matched_1[i][1], \
              list_matched_2[i][0], list_matched_2[i][1]) )

# reading FITS files
(header1, image1) = read_fits (file_fits1)
(header2, image2) = read_fits (file_fits2)

# byte swap
# data stored in FITS file is network byte-order (big endian).
# Intel/AMD CPUs use little endian.
image1 = image1.byteswap ().newbyteorder ()
image2 = image2.byteswap ().newbyteorder ()

# aligning 2nd image to 1st image
st = skimage.transform.SimilarityTransform (scale=transf.scale, \
                                            rotation=transf.rotation, \
                                            translation=transf.translation)
image2_aligned = skimage.transform.warp (image2, st.inverse)

# marker and colour names for matplotlib
markers = ['o', 'v', '^', 's', 'p', 'h', '8']
colours = ['maroon', 'red', 'coral', 'bisque', 'orange', \
           'wheat', 'yellow', 'green', 'lime', 'aqua', \
           'skyblue', 'blue', 'indigo', 'violet', 'pink']

# 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 first image
norm1 \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image1) )
im1 = ax1.imshow (image1, origin='lower', cmap='bone', norm=norm1)
for i in range ( len (list_matched_1) ):
    i_marker = i % len (markers)
    i_colour = i % len (colours)
    ax1.plot (list_matched_1[i][0], list_matched_1[i][1], \
              marker=markers[i_marker], color=colours[i_colour], \
              markersize=8, fillstyle='none')
ax1.set_title ('First Image')

# plotting second image
norm2 \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image2_aligned) )
im2 = ax2.imshow (image2_aligned, origin='lower', cmap='bone', norm=norm2)
for i in range ( len (list_matched_2_aligned) ):
    i_marker = i % len (markers)
    i_colour = i % len (colours)
    ax2.plot (list_matched_2_aligned[i][0], list_matched_2_aligned[i][1], \
              marker=markers[i_marker], color=colours[i_colour], \
              markersize=8, fillstyle='none')
ax2.set_title ('Second Image')

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

Run the script.

```
% chmod a+x advobs202202_s17_07.py
% ./advobs202202_s17_07.py -h
usage: advobs202202_s17_07.py [-h] [-o FILE_OUTPUT]
                              catalogue1 catalogue2 fits1 fits2

aligning image

positional arguments:
  catalogue1            catalogue file 1
  catalogue2            catalogue file 2
  fits1                 FITS file 1
  fits2                 FITS file 2

optional arguments:
  -h, --help            show this help message and exit
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output figure file

% ./advobs202202_s17_07.py -o match_1.pdf stars_0.cat stars_1.cat \
? stars_0.fits stars_1.fits
#
# result of image alignment
#
#   date/time = 2022-06-09 22:54:06.910543
#
# input files
#
#   catalogue file 1 = stars_0.cat
#   catalogue file 2 = stars_1.cat
#
# transformation matrix
#
# [
#   [0.965930, 0.258819, -291.031611],
#   [-0.258819, 0.965930, 285.168282],
#   [0.000000, 0.000000, 1.000000]
# ]
#
#
# list of matched stars
#
( 1866.8354,  1690.2168) on 1st image ==> ( 1720.6830,  1915.6673) on 2nd image
( 1077.5368,   318.7460) on 1st image ==> ( 1313.2339,   386.6019) on 2nd image
( 1067.4470,   801.2203) on 1st image ==> ( 1178.6199,   850.0304) on 2nd image
( 1148.1532,  1002.5993) on 1st image ==> ( 1204.4642,  1065.4658) on 2nd image
( 1754.8088,  1644.8986) on 1st image ==> ( 1624.1923,  1842.8884) on 2nd image
(  631.2147,   186.6385) on 1st image ==> (  916.3337,   143.5095) on 2nd image
( 1579.5159,    79.6083) on 1st image ==> ( 1860.0076,   285.5751) on 2nd image
(  333.5968,   757.1984) on 1st image ==> (  481.1349,   617.5982) on 2nd image
( 1564.1786,   545.0747) on 1st image ==> ( 1724.6907,   731.2205) on 2nd image
(  175.0280,   516.6916) on 1st image ==> (  390.2882,   344.3251) on 2nd image
(  531.4456,  1700.6104) on 1st image ==> (  428.1424,  1580.0668) on 2nd image
(  456.2789,  1283.4888) on 1st image ==> (  463.4521,  1157.7099) on 2nd image
( 1484.5031,  1835.8843) on 1st image ==> ( 1313.6694,  1957.4054) on 2nd image
( 1802.5422,  1469.1342) on 1st image ==> ( 1715.7960,  1685.4752) on 2nd image
(  490.6636,   523.7812) on 1st image ==> (  693.2843,   432.8002) on 2nd image
(  183.3418,  1942.4515) on 1st image ==> (   29.2805,  1723.6011) on 2nd image
(  207.9840,   853.1035) on 1st image ==> (  335.0328,   677.7534) on 2nd image
(  420.5189,   650.0014) on 1st image ==> (  592.8722,   536.5421) on 2nd image
( 1546.6757,  1774.0108) on 1st image ==> ( 1389.7549,  1913.7522) on 2nd image
( 1709.5695,  1240.9216) on 1st image ==> ( 1685.0338,  1441.0519) on 2nd image
(  314.8656,   528.8662) on 1st image ==> (  522.1874,   392.2157) on 2nd image
(  161.1666,  1839.5355) on 1st image ==> (   34.5014,  1618.4186) on 2nd image
(   81.7757,   742.0294) on 1st image ==> (  241.8532,   537.7799) on 2nd image
(  756.3128,  1269.0433) on 1st image ==> (  757.0154,  1221.4139) on 2nd image
( 1109.6329,   996.5852) on 1st image ==> ( 1168.8099,  1049.6668) on 2nd image
( 1608.5440,   240.8841) on 1st image ==> ( 1846.2962,   448.8502) on 2nd image
( 1284.6825,  1062.6569) on 1st image ==> ( 1320.8116,  1158.8328) on 2nd image
% ls -lF match_*.pdf
-rw-r--r--  1 daisuke  taiwan  427388 Jun  9 22:46 match_0.pdf
-rw-r--r--  1 daisuke  taiwan  422065 Jun  9 22:54 match_1.pdf
```

Display the PDF file, and check the result. (Fig. 10)

```
% okular match_1.pdf
```

8 Dealing with real data

Next, we deal with real data.

8.1 Downloading SDSS image

Make a Python script to download DSS/SDSS images.

Python Code 8: advobs202202_s17_08.py

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

#
# Time-stamp: <2022/06/09 23:03:23 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing astroquery module
import astroquery.simbad
import astroquery.ipac.ned
import astroquery.skyview
```

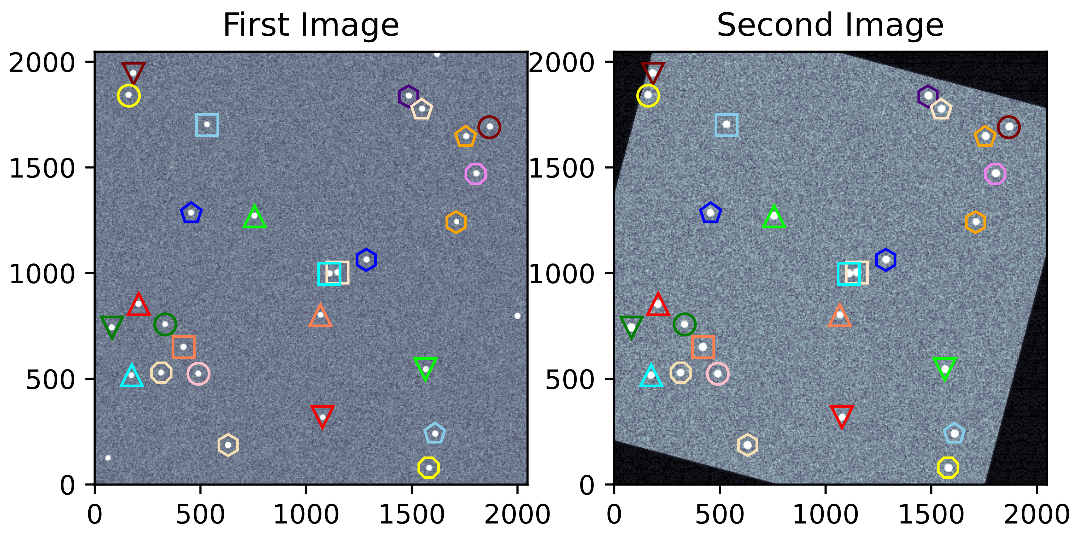


Figure 10: Result of finding star-to-star correspondence. The 2nd image is aligned to the 1st image.

```
# importing astropy module
import astropy.coordinates
import astropy.units

# importing datetime module
import datetime

# importing ssl module
import ssl

# allow insecure downloading
ssl._create_default_https_context = ssl._create_unverified_context

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

# units
u_ha  = astropy.units.hourangle
u_deg = astropy.units.deg

# constructing parser object
desc = "downloading DSS/SDSS image"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
list_resolver = ['simbad', 'ned']
list_survey   = ['DSS1 Blue', 'DSS1 Red', 'DSS2 Blue', 'DSS2 Red', 'DSS2 IR', \
                 'SDSSu', 'SDSSg', 'SDSSr', 'SDSSi', 'SDSSz']
parser.add_argument ('-r', '--resolver', choices=list_resolver, \
                     default='simbad', help='choice of name resolver')
parser.add_argument ('-s', '--survey', choices=list_survey, \
                     default='SDSSr', help='choice of survey')
parser.add_argument ('-t', '--target', default='', help='target name')
parser.add_argument ('-f', '--fov', type=int, default=1024, \
                     help='field-of-view in pixel')
parser.add_argument ('-o', '--output', default='', help='output file name')

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

# input parameters
name_resolver = args.resolver
survey        = args.survey
target_name   = args.target
fov_pix       = args.fov
file_output   = args.output

# checking target name
if (target_name == ''):
    # printing error message
    print ("No target name is given!")
    # exit
    sys.exit ()

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

# checking output file name
```

```
if (file_output == ''):
    # printing error message
    print ("No output file name is given!")
    # exit
    sys.exit ()
elif not (path_output.suffix == '.fits'):
    # printing error message
    print ("Output file must be FITS file!")
    # exit
    sys.exit ()
if (path_output.exists ()):
    # printing error message
    print ("Output file '%s' exists!" % file_output)
    # exit
    sys.exit ()

# using name resolver
if (name_resolver == 'simbad'):
    query_result = astroquery.simbad.Simbad.query_object (target_name)
elif (name_resolver == 'ned'):
    query_result = astroquery.ipac.ned.Ned.query_object (target_name)

# RA and Dec
RA  = query_result['RA']
Dec = query_result['DEC']

# coordinate
if (name_resolver == 'simbad'):
    coord = astropy.coordinates.SkyCoord (RA[0], Dec[0], unit=(u_ha, u_deg))
elif (name_resolver == 'ned'):
    coord = astropy.coordinates.SkyCoord (RA[0], Dec[0], unit=(u_deg, u_deg))

coord_str = coord.to_string (style='hmsdms')
(coord_ra_str, coord_dec_str) = coord_str.split ()
coord_ra_deg  = coord.ra.deg
coord_dec_deg = coord.dec.deg

# printing coordinate
print ("Target Name: %s" % target_name)
print ("  RA:  %s = %f deg" % (coord_ra_str, coord_ra_deg) )
print ("  Dec: %s = %f deg" % (coord_dec_str, coord_dec_deg) )

# searching image
list_image = astroquery.skyview.SkyView.get_image_list (position=coord, \
                                                         survey=survey)

# printing image list
print ("Available images:")
print (" ", list_image)

# getting image
image = astroquery.skyview.SkyView.get_images (position=coord, survey=survey, \
                                               pixels=fov_pix)

# header and data
image0 = image[0]
header = image0[0].header
data   = image0[0].data
```

```
# adding comments in header
header['history'] = "image downloaded from %s" % survey
header['history'] = "image saved on %s" % now

# saving to a FITS file
astropy.io.fits.writeto (file_output, data, header=header)
```

Execute the script, and download SDSS image.

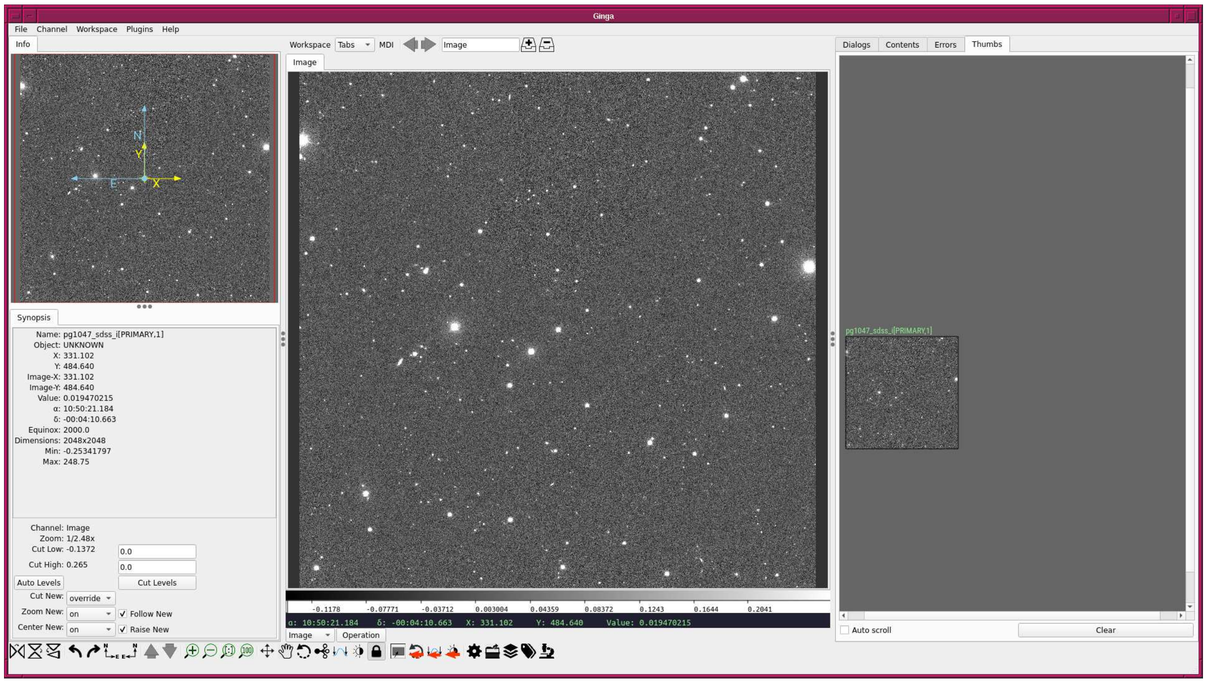
```
% chmod a+x advobs202202_s17_08.py
% ./advobs202202_s17_08.py -h
usage: advobs202202_s17_08.py [-h] [-r {simbad,ned}]
                              [-s {DSS1 Blue,DSS1 Red,DSS2 Blue,DSS2 Red,DSS2 IR
,SDSSu,SDSSg,SDSSr,SDSSi,SDSSz}]
                              [-t TARGET] [-f FOV] [-o OUTPUT]

downloading DSS/SDSS image

optional arguments:
  -h, --help            show this help message and exit
  -r {simbad,ned}, --resolver {simbad,ned}
                        choice of name resolver
  -s {DSS1 Blue,DSS1 Red,DSS2 Blue,DSS2 Red,DSS2 IR,SDSSu,SDSSg,SDSSr,SDSSi,SDSS
z}, --survey {DSS1 Blue,DSS1 Red,DSS2 Blue,DSS2 Red,DSS2 IR,SDSSu,SDSSg,SDSSr,SD
SSi,SDSSz}
                        choice of survey
  -t TARGET, --target TARGET
                        target name
  -f FOV, --fov FOV     field-of-view in pixel
  -o OUTPUT, --output OUTPUT
                        output file name

% ./advobs202202_s17_08.py -s SDSSi -t PG1047+003 -f 2048 -o pg1047_sdss_i.fits
Target Name: PG1047+003
  RA:  10h50m02.8265s = 162.511777 deg
  Dec: -00d00m36.879s = -0.010244 deg
Available images:
  ['https://skyview.gsfc.nasa.gov/tempspace/fits/skv9049887459538.fits']
Downloading https://skyview.gsfc.nasa.gov/tempspace/fits/skv9049893700664.fits
|=========================================|  16M/ 16M (100.00%)        29s
% ls -lF *.fits
-rw-r--r--  1 daisuke  taiwan  16793280 Jun  9 23:07 pg1047_sdss_i.fits
-rw-r--r--  1 daisuke  taiwan  33560640 Jun  9 22:08 stars_0.fits
-rw-r--r--  1 daisuke  taiwan  33560640 Jun  9 22:09 stars_1.fits
```

Use Ginga to check the image. (Fig. 11)

```
% ginga pg1047_sdss_i.fits &
```

8.2 Downloading DSS image

Download DSS image.



```
Name: pg1047_sdss_i[PRIMARY,1]
Object: UNKNOWN
X: 331.102
Y: 484.640
Image-X: 331.102
Image-Y: 484.640
Value: 0.019470215
α: 10:50:21.184
δ: -00:04:10.663
Equinox: 2000.0
Dimensions: 2048x2048
Min: -0.25341797
Max: 248.75

Channel: Image
Zoom: 1/2.48x
Cut Low: -0.1372     0.0
Cut High: 0.265      0.0
Cut New: override
Zoom New: on         ✓ Follow New
Center New: on       ✓ Raise New

-0.1178  -0.07771  -0.03712  0.003004  0.04359  0.08372  0.1243  0.1644  0.2041
α: 10:50:21.184   δ: -00:04:10.663   X: 331.102   Y: 484.640   Value: 0.019470215
```

Figure 11: SDSS i'-band image of PG1047+003.

```
% ./advobs202202_s17_08.py -s "DSS2 Blue" -t PG1047+003 -f 768 \
? -o pg1047_dss_b.fits
Target Name: PG1047+003
  RA:  10h50m02.8265s = 162.511777 deg
  Dec: -00d00m36.879s = -0.010244 deg
Available images:
  ['https://skyview.gsfc.nasa.gov/tempspace/fits/skv9050301508222.fits']
Downloading https://skyview.gsfc.nasa.gov/tempspace/fits/skv9050302882337.fits
|=========================================| 2.3M/2.3M (100.00%)         5s
% ls -lF pg1047_*.fits
-rw-r--r--  1 daisuke  taiwan   2373120 Jun  9 23:13 pg1047_dss_b.fits
-rw-r--r--  1 daisuke  taiwan  16793280 Jun  9 23:07 pg1047_sdss_i.fits
```

Use Ginga to check the image. (Fig. 12)

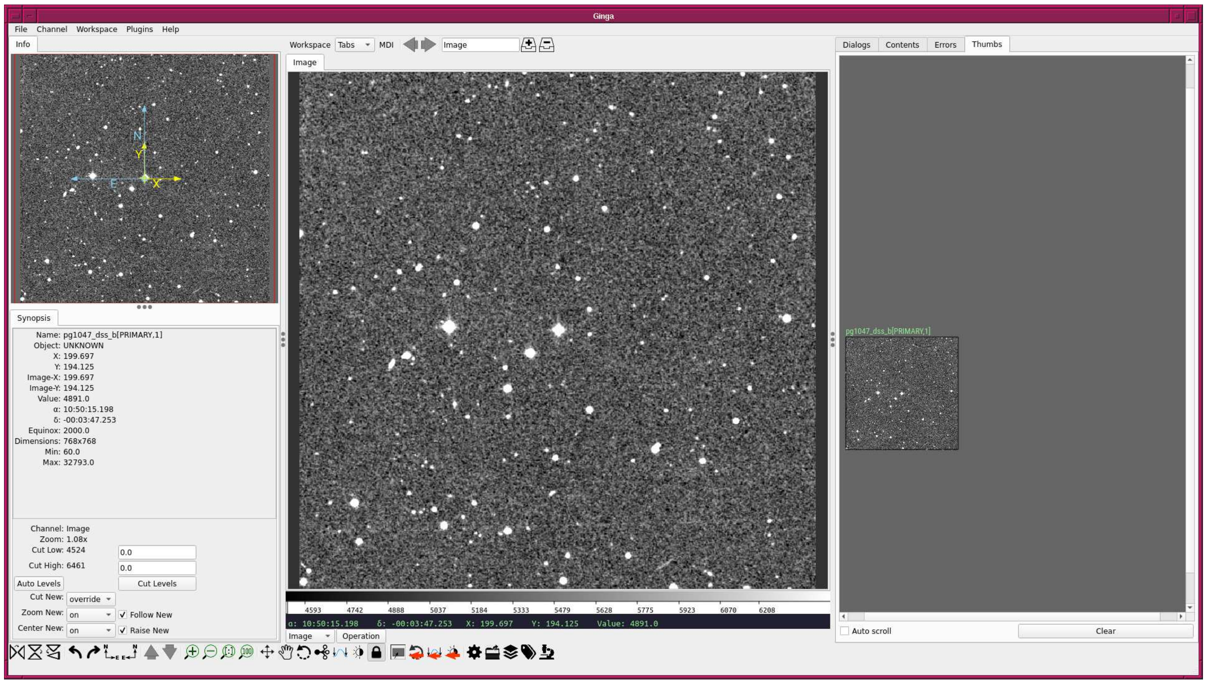
```
% ginga pg1047_dss_b.fits &
```

8.3 Source extraction

Carry out source extraction for both SDSS and DSS images.

```
% ./advobs202202_s17_04.py -i pg1047_sdss_i.fits -t 100.0 -o pg1047_sdss_i.cat \
? -g pg1047_sdss_i.pdf
% ./advobs202202_s17_04.py -i pg1047_dss_b.fits -t 30.0 -o pg1047_dss_b.cat \
? -g pg1047_dss_b.pdf
% ls -lF pg1047_*.cat
-rw-r--r--  1 daisuke  taiwan   7051 Jun  9 23:16 pg1047_dss_b.cat
-rw-r--r--  1 daisuke  taiwan  11357 Jun  9 23:16 pg1047_sdss_i.cat
```

Results of source extraction are shown in Fig. 13 and 14.



```
       Name: pg1047_dss_b[PRIMARY,1]
     Object: UNKNOWN
          X: 199.697
          Y: 194.125
    Image-X: 199.697
    Image-Y: 194.125
      Value: 4891.0
          α: 10:50:15.198
          δ: -00:03:47.253
   Equinox: 2000.0
Dimensions: 768x768
        Min: 60.0
        Max: 32793.0

    Channel: Image
       Zoom: 1.08x
    Cut Low: 4524        0.0
   Cut High: 6461        0.0
Auto Levels              Cut Levels
    Cut New: override
   Zoom New: on          ✓ Follow New
 Center New: on          ✓ Raise New

α: 10:50:15.198   δ: -00:03:47.253   X: 199.697   Y: 194.125   Value: 4891.0
```

Figure 12: DSS Blue image of PG1047+003.

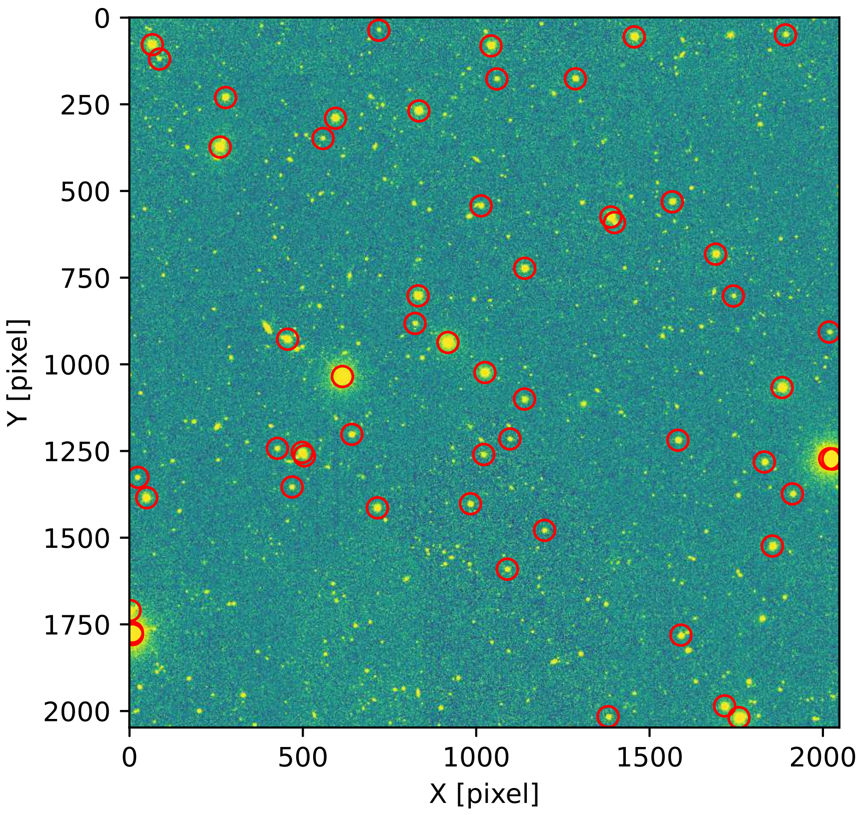


Figure 13: Result of source extraction of SDSS image.

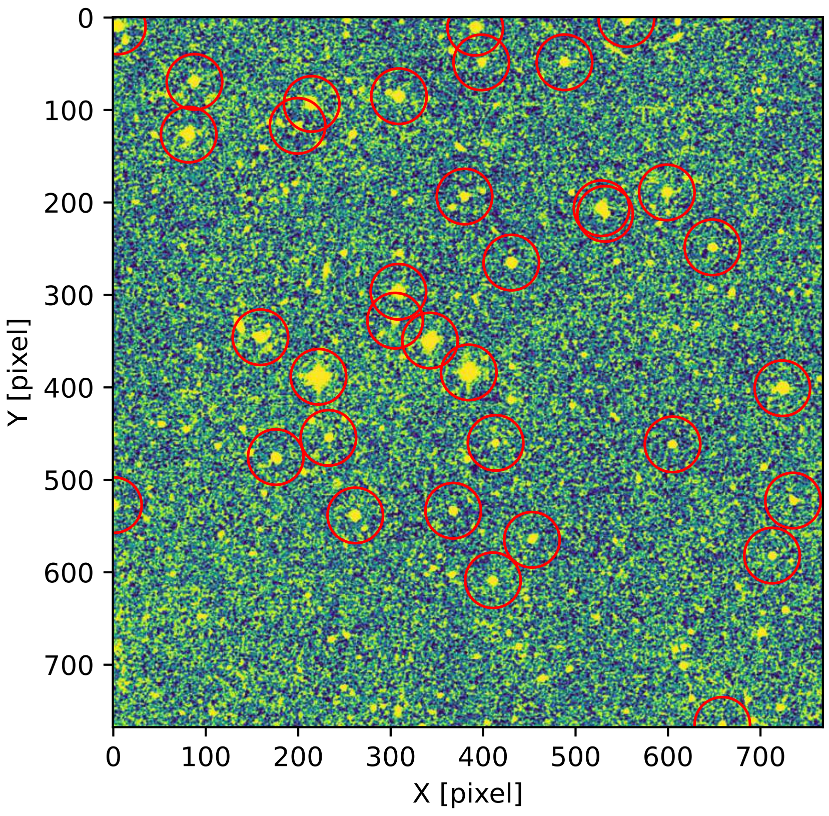


Figure 14: Result of source extraction of DSS image.

8.4 Image alignment

Carry out star-to-star matching.

```
% ./advobs202202_s17_06.py -o match_2.pdf pg1047_sdss_i.cat pg1047_dss_b.cat \
? pg1047_sdss_i.fits pg1047_dss_b.fits
#
# result of image alignment
#
#   date/time = 2022-06-09 23:18:21.691987
#
# input files
#
#   catalogue file 1 = pg1047_sdss_i.cat
#   catalogue file 2 = pg1047_dss_b.cat
#
# transformation matrix
#
# [
#   [2.525202, -0.000804, 54.395081],
#   [0.000804, 2.525202, 53.794411],
#   [0.000000, 0.000000, 1.000000]
# ]
#
#
# list of matched stars
#
(  823.6800,   882.0045) on 1st image ==> (  304.8340,   327.8275) on 2nd image
(  498.3468,  1253.8589) on 1st image ==> (  175.7000,   475.3944) on 2nd image
( 1059.4451,   176.9533) on 1st image ==> (  398.0028,    48.5176) on 2nd image
( 1398.8033,   591.1109) on 1st image ==> (  531.8952,   212.3450) on 2nd image
( 1286.1954,   176.4987) on 1st image ==> (  487.9928,    48.3692) on 2nd image
(  832.3487,   802.4233) on 1st image ==> (  308.3825,   296.5177) on 2nd image
( 1581.7806,  1218.8741) on 1st image ==> (  604.8763,   461.6491) on 2nd image
(  918.5140,   936.9002) on 1st image ==> (  342.5177,   349.3132) on 2nd image
(   65.7829,    77.9935) on 1st image ==> (    4.9078,     9.6261) on 2nd image
( 1013.9307,   542.8151) on 1st image ==> (  379.8766,   193.5695) on 2nd image
(  715.1062,  1413.9673) on 1st image ==> (  261.7456,   538.4762) on 2nd image
( 1025.2500,  1023.6115) on 1st image ==> (  384.4590,   383.7996) on 2nd image
( 1716.5243,  1985.1198) on 1st image ==> (  658.6876,   764.9825) on 2nd image
(  558.1739,   348.6601) on 1st image ==> (  199.3383,   117.1204) on 2nd image
( 1197.0451,  1478.8071) on 1st image ==> (  452.8383,   564.8611) on 2nd image
( 1042.7106,    81.4663) on 1st image ==> (  391.4883,    11.0018) on 2nd image
(  614.1969,  1035.1081) on 1st image ==> (  221.9201,   388.4945) on 2nd image
( 1854.1293,  1524.1565) on 1st image ==> (  712.4975,   581.9372) on 2nd image
( 1911.9819,  1373.6673) on 1st image ==> (  735.2345,   522.4108) on 2nd image
( 1690.5344,   682.2595) on 1st image ==> (  647.9930,   248.5211) on 2nd image
(  641.1427,  1202.0095) on 1st image ==> (  232.6394,   454.6186) on 2nd image
(  277.2246,   230.5123) on 1st image ==> (   87.7854,    69.6134) on 2nd image
( 1882.3096,  1066.9590) on 1st image ==> (  723.6902,   400.9431) on 2nd image
( 1097.2289,  1215.2369) on 1st image ==> (  413.5035,   460.0289) on 2nd image
(  261.4060,   373.2712) on 1st image ==> (   81.5038,   126.6262) on 2nd image
( 1565.4674,   531.2258) on 1st image ==> (  598.6880,   189.1224) on 2nd image
( 1089.7507,  1590.7803) on 1st image ==> (  410.5949,   608.6447) on 2nd image
(  456.1203,   927.5883) on 1st image ==> (  159.1240,   345.7897) on 2nd image
( 1388.3806,   575.0339) on 1st image ==> (  528.1273,   206.2133) on 2nd image
( 1139.8708,   723.0177) on 1st image ==> (  430.4985,   265.0112) on 2nd image
(  834.7360,   269.1596) on 1st image ==> (  308.9729,    85.0933) on 2nd image
(  983.2979,  1402.0423) on 1st image ==> (  367.6290,   533.3973) on 2nd image
```

```
% ls -lF match_?.pdf
-rw-r--r--  1 daisuke  taiwan  427388 Jun  9 22:46 match_0.pdf
-rw-r--r--  1 daisuke  taiwan  422065 Jun  9 22:54 match_1.pdf
-rw-r--r--  1 daisuke  taiwan  477073 Jun  9 23:18 match_2.pdf
```

Display the PDF file. (Fig. 15)

```
% xpdf match_2.pdf
```

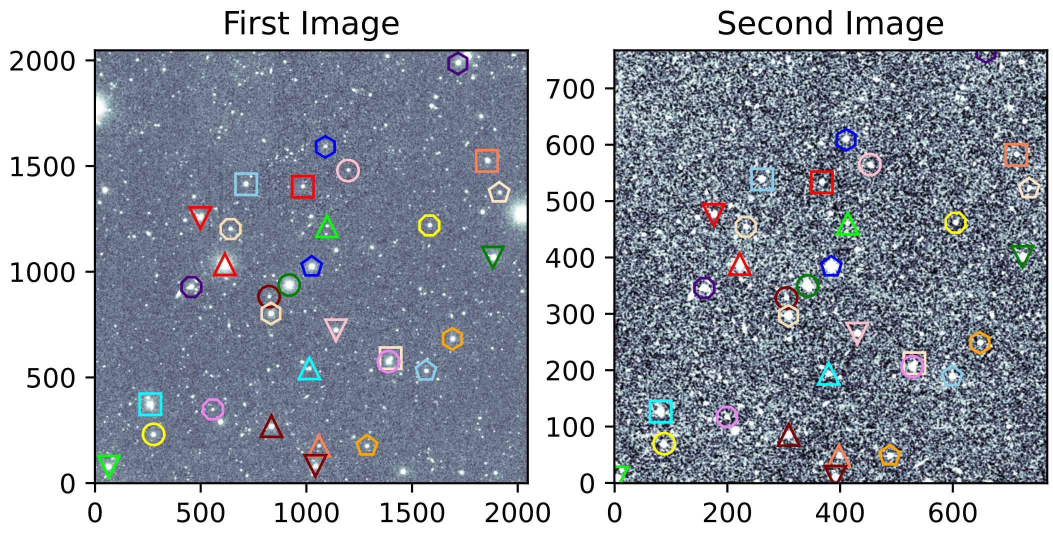


Figure 15: The star-to-star correspondence between SDSS and DSS images.

8.5 Aligning SDSS g' and r' band images

Download SDSS g' and r' band images of the same region of the sky. Align those images.

8.6 Aligning optical image and NIR image

Download an optical image and a NIR image of the same region of the sky. Align those images.

9 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`
7. "astroalign"
   - `https://astroalign.readthedocs.io/`
8. "scikit-image"
   - `https://scikit-image.org/`
9. "astroquery"
   - `https://astroquery.readthedocs.io/`

10 Exercises

1. Describe mathematical expression of translation of 2-dimensional coordinates (x, y).
2. Describe mathematical expression of rotation of 2-dimensional coordinates (x, y).
3. Choose one r'-band and i'-band image of SDSS images. Make your own Python script to try image alignment. Show the source code of your Python script. Show the result of your star-to-star correspondence and image alignment. What is the transformation matrix?
4. Choose one g'-band image of Lulin One-meter Telescope data. Choose one r'-band image of Lulin One-meter Telescoep data of the same field as g'-band data. Make your own Python script to try image alignment. Show the source code of your Python script. Show the result of your star-to-star correspondence and image alignment. What is the transformation matrix?
5. Choose one g'-band image of Lulin One-meter Telescope data. Download SDSS image of the same field. Make your own Python script to try image alignment. Show the source code of your Python script. Show the result of your star-to-star correspondence and image alignment. What is the transformation matrix?