# Tutorial VII: Advanced Image Analysis

PHY 499/5590 – Fundamentals of Astronomy Research

Kameswara Bharadwaj Mantha & Daniel H. McIntosh
Univ. of Missouri Kansas City, Spring 2020.
Due Date: April 20, 2020

## Purpose & Expectations

In this tutorial, you will expand your knowledge of Python 2D image analysis. You will learn to manipulate images (e.g., making cutouts from a larger mosaic), accessing image data and performing quantitative analysis (such as photometry). You will also learn some image aesthetics (e.g., showing markers on an image).

Please answer all the tasks and activities throughout the tutorial and attach relevant python codes along with your presentation PDF.

Collaborative problem solving is encouraged, however, you must give full disclosure if you seek help from your fellow peers or from the internet. Plagiarism, directly copying work from the internet without showcasing effort to learn and understand is strictly discouraged and such actions will be reported.

# 1 Cutouts of Images

One of the handy tools that you should get experienced with is to make postage stamps of objects from a larger image. These postage stamps are advantageous for many reasons such as showing individual objects of interest, running additional software on them (which is much faster than running it on a large image), etc. In this section, let's look at an easy way to make cutouts and then save the resultant images as a separate fits file. These steps will include a lot of small (yet important steps) that you will use independently in other tasks.

One can start by loading or opening a larger image using your previous image visualization knowledge as following:

```
large_image = fits.open(file_name)
```

As I mentioned previously, you can access the header and the image data as:

```
large_image_data = large_image[0].data
large_image_header = large_image[0].header
```

A shortcut for the above steps is to directly load the data and the header, without actually having to "open" the fits file. This can be done as the following:

```
large_image_data = fits.getdata(file_name)
large_image_header = fits.getheader(file_name)
```

Next, you would need the "world coordinate system" (wcs) to map the pixels in the image to the sky. For this you need to import the module "wcs" as:

```
from astropy.wcs import WCS
```

Then, you can use the header information to generate the image's wcs information as follows:

```
large_image_wcs = WCS(large_image_header)
```

The image data is a 2-dimensional array and we want to cut a rectangular (or a square) portion of it. The two obvious parameters one would need to know are the "center" at which you wish to make the cutout of and the "size" of the postage stamp (in pixels). You can use the "Cutout2D" module in `astropy` to make the cutouts as following:

```
from astropy.nddata import Cutout2D
cutout_image = Cutout2D(large_image_data,(x_cen,y_cen),(xsize, ysize),wcs =
    large_image_wcs)
```

You can access the data and wcs information of the cutout you made as following:

```
cutout_data = cutout_image.data
cutout_wcs = cutout_image.wcs
```

Note that the world coordinate system of the cutout is going to be different from that of the larger image. Therefore, it is key to retrieve the new wcs and then use this to save a new header when writing the cutout information into a new fits file. You can write the cutout information as a fits file as:

```
fits.writeto(new_file_name,cutout_data, header=cutout_wcs.to_header())
```

## TASK-I

Using the information provided above, please do the following exercise.

1. Using the large_mosaic.fits image in your data folder, make a 200×200 postage stamp centered on $(505, 506)$. Save the cutout information as a separate fits file. Show the cutout using Python with optimal visualization. Your figure should follow the thumb rules of professional plotting.

2. Run source extraction (with `sep`) using the CONFIG 2 of your previous tutorial on the large_mosaic.fits. Make (100,100) sized cutouts of 5 randomly selected objects from the image. Note: You can use the `np.random.choice(objects,number_of_objects)` to randomly select objects, where `np` stands for `numpy` imported as such. For any hints look at `https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.choice.html`.

# Visualization of Markers on Images

In professional talks and papers, you will often see different markers (e.g., circles or ellipses) overlaid on top of the images to illustrate some point. This is an important skill. Let's see how you can easily do this using Python. For this purpose, you will use the `Ellipse and Circle` modules within `matplotlib` as follows:

```
from matplotlib.patches import Ellipse, Circle
```

If you know where to place the marker (e.g., center) and their size (e.g., radius), you can show a circle and ellipse as:

```
c = Circle(xy=(xcenter,ycenter),radius=your_radius)
e = Ellipse(xy=(xcenter,ycenter), width=your_width,height=your_height,angle=
    your_angle)
```

For aesthetic purposes, we would want to make the marker edge-only, for example as follows:

```
c.set_facecolor('none')
c.set_edgecolor('red')
```

This just creates the circle, but doesn't plot it. Therefore, we have to explicitly add the marker onto the plot as follows:

```
axis.add_artist(c)
axis.add_artist(e)
```

## TASK-II

Using the information provided to you on visualizing markers on images, please do the following exercise.

1. Using the $200 \times 200$ postage stamp created in your previous task, please run source extraction (using `sep`) with your configuration of choice. Visualize the image with optimal scaling and then plot circles (with radius of 7 pixels) on each detected object.

2. Instead of circles on each object, use the information from source extraction to plot ellipses on each object. The width and height of each ellipse should be 6 times the semi-major ($a$) and semi-minor ($b$) axis of each object, respectively. Note: see the website `https://sep.readthedocs.io/en/v1.0.x/tutorial.html` for hint.

## 1.1 Masking using Segmentation Regions

In some applications, you may want to mask (or unmask) regions that correspond to a specific region or object. One of the easiest ways to do this is use the segmentation map generated during your source extraction process and use it to your advantage.

I will demonstrate how to mask pixels corresponding to a specific region within the segmentation map. For example, I want to mask the object labelled "1" in the segmentation map. Here is how I would do it:

```
mask = deepcopy(segmentation_map)
mask[np.where(mask ==1)] = 0
masked_data = mask * your_image_data
```

Note that `deepcopy` will copy the information given to it into a separate memory instance to prevent any permanent changes to `segmentation map`. Make sure to import deepcopy as: "`from copy import deepcopy`" wherever needed.

## TASK-III

1. Please read the above code block and conceptually explain each step.

2. Using the $200 \times 200$ cutout made during your previous task, perform source extraction and using the segmentation region, mask the source "2" and make a four panel figure, where the first panel is the cutout image, second panel is the segmentation map, third panel is the segmentation map with the 2nd source masked, and the fourth panel is cutout image (shown in the first panel) with the 2nd source masked.

3. If you were given a segmentation map, please make an algorithm flow chart on how you will create a new image where all the pixels corresponding to an object are made ZERO and the background is made ONE.

4. Using your algorithm, please write a code that will generate a binary image with all sources masked ($= 0$). Show your steps in a four panel figure made with Python: the first panel is the cutout image, second panel is the segmentation map, third panel is the segmentation map with the all sources masked, and the fourth panel is cutout image (shown in the first panel) with the all sources masked. Your plot should look something like Figure 1.

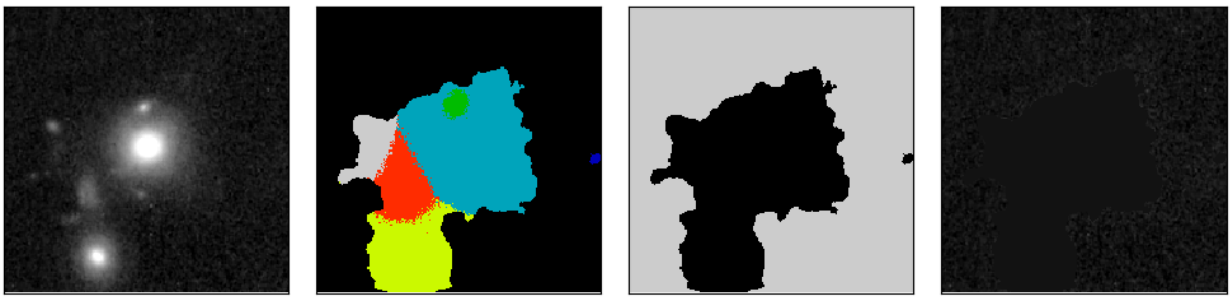

Figure 1: Please try to match this figure for TASK-III, item 4.

# 2 Quantitative Analysis of the Image Data

One of the key steps during Image Analysis is to be able to make quantitative measures (such as flux) from objects by counting up the amount of light in all pixels corresponding to them. In this section, we will look at how to access and quantify the image data.

## 2.1 Local Background Sky Distributions

A straightforward example from what we have covered so far is to quantify the distribution of pixel values corresponding to the background sky (i.e., everything except for the sources). If your object masked data is given the variable "**object_masked_data**", then can plot the histogram as:

```
for_histogram = [i for i in object_masked_data.flat if i!=0]
histogram, bins_edges, something = plt.hist(for_histogram, bins=number_of_bins
    ,density=True, histtype='step')
```

Note, that I am using ".flat" word, which "flattens" a 2-dimensional array into a 1D array. Also, the values of pixels are stored in units of `electrons per second`.

It is often the case that you would want to fit this histogram with a Gaussian (bell curve) and then retrieve the mean and standard deviation of it. Given that you recollect how to compute bin centers from bin edges, you can fit the histogram as such:

```
from astropy.modeling import models, fitting
guess_mean = np.mean(for_histogram)
guess_std = np.std(for_histogram)
guess_amplitude = np.max(for_histogram)

gaussian = models.Gaussian1D(amplitude=guess_amplitude,mean=guess_mean, stddev
    =guess_std)

fitter = fitting.LevMarLSQFitter()
gaussian_fit = fitter(gaussian, bin_centers, for_histogram)
```

Once you have fitted the histogram with a Gaussian curve, you would want to show the best-fit curve on top of the histogram. You can do so as:

```
x_array = np.arange(np.min(for_histogram),np.max(for_histogram),0.001)
y_array = gaussian_fit(x_array)
plt.plot(x_array,y_array,linestyle='--',color='red',marker='None',label='
    Gaussian Fit')
```

You can also query the best-fit mean and standard deviation of the Gaussian curve by doing:

```
best_fit_mean = gaussian_fit.mean.value
best_fit_std = gaussian_fit.stddev.value
```

## TASK-IV

Synthesize the above information on local-sky background distribution and answer the following questions.

1. Please explain the above code blocks (line-by-line) in a conceptual fashion.
2. Please make a professional plot (properly labelled axis and fontsize) of the distribution of background sky pixels.
3. Please fit the histogram data with a Gaussian. In text, please print the best-fit mean and standard deviation. Try to match Figure 2.

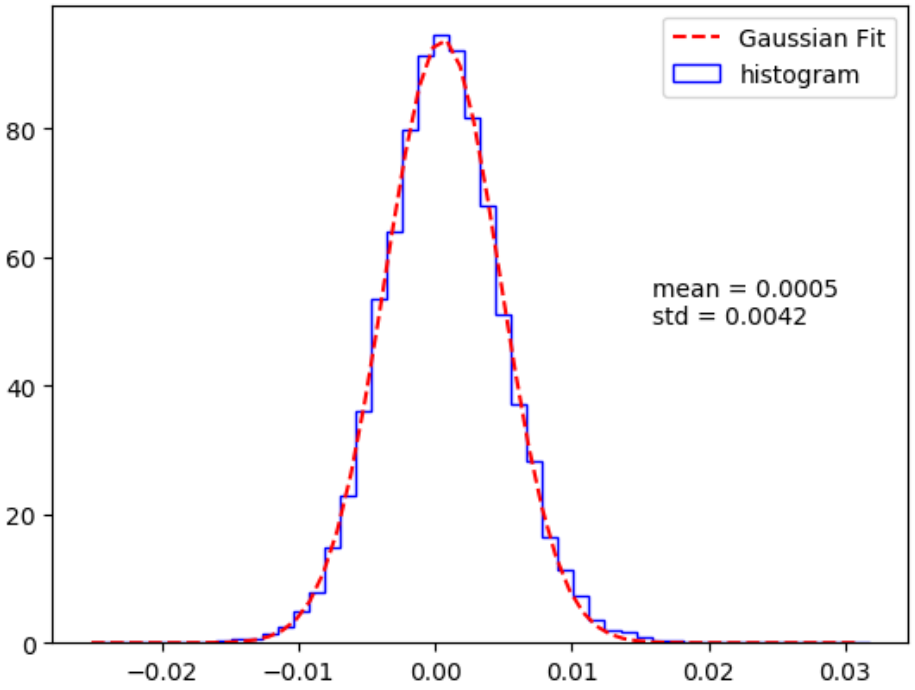


Figure 2: Try to match this figure for item 3 of Task-IV

## 2.2 Aperture Photometry

Photometry (measurement of light) lies at the crux of every astro-related analysis. Let's try to see how we can perform some simple photometry with Python. For this purpose, you will use the `photutils package`. Conceptually, we would want to measure the light from an object within an aperture (of a certain shape), the most common aperture shape is a Circle, with some user defined size. Assuming that you ran source extraction with `sep` and the information of the extracted objects is stored in the variable **objects**, let's generate a list of tuples with the (x,y) positions of objects:

```
positions = [(i, j) for i, j in zip(objects['x'], objects['y'])]
```

Now, lets create circular apertures with a radius 10 pixels, centered at the positions above:

```
apertures = CircularAperture(positions, r=10.)
```

Finally, lets measure the flux in each aperture as:

```
phot_table = aperture_photometry(bkg_subtracted, apertures)
```

If you wish to show the apertures overlaid on top of the image, then you can do:

```
apertures.plot(color='red', lw=1.5, alpha=0.5,ls='--')
```

## TASK-V

1. Please explain conceptually each line of the aperture photometry code blocks.

2. Using the cutout made in the previous tasks, run source extraction with `sep` and then perform aperture photometry on all the detected objects in the cutout. Make a figure showing the image in proper visualization and overlay the apertures on top of it. Try to match Figure 3.

3. Print the photometry table. Assuming that the units in which image information is stored is electron/second, compute the flux in micro-Janskies ($10^{-6}$Jy) for the brightest object in the image. Assume the calibration factor is $1.5 \times 10^{-7}$Jy sec/electron.

4. Assuming that the magnitude zero point (zp) is 25.96, convert the flux of the brightest object in your table to apparent magnitudes ($m = -2.5 \log_{10}(f) + zp$).

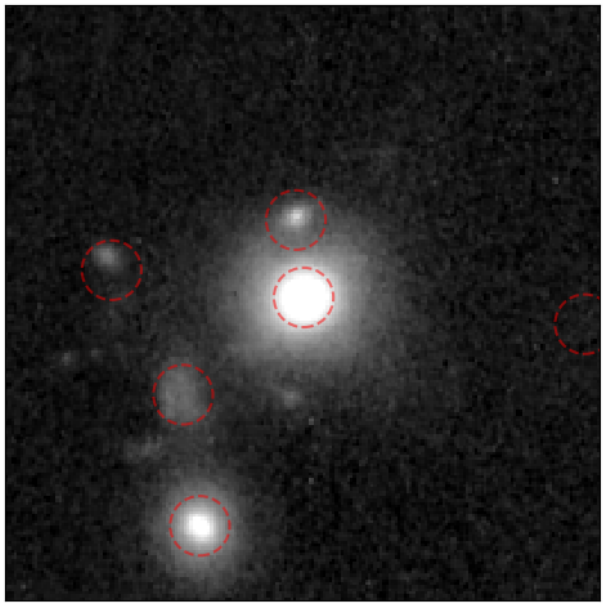

Figure 3: Please try to match this figure for item 2 of Task-V.

## 2.3 Segmentation Map based Flux Measurement

Another interesting way to count up the flux corresponding a specific object is using the source extraction based segmentation map. Conceptually speaking, we will need to make a mask such that everything except the object of interest should be set to zeros and the object itself should be marked with ones. This binary image, when multiplied by the actual image data, will only have the pixels corresponding to that object.

First, lets create a mask with every pixel corresponding to an object number of interest (**object_number**) set to zeros:

```
mask = deepcopy(segmap)
mask[np.where(mask != object_number)] = 0
```

Next, since the pixel values in the segmentation map can range between 1 and above (depending on the object number), we need to change these to ones by:

```
mask[np.where(mask>1)] = 1
```

Once you have created this mask, you can multipy this with the image data and them sum it.

```
flux = np.sum(...) # np is numpy imported as such
```

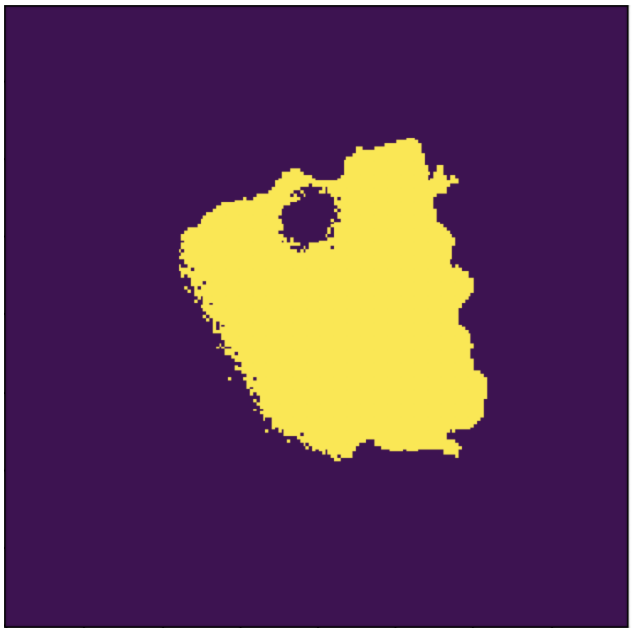

Figure 4: This is how your mask should look like, where the purple regions have a value of zero and yellow corresponds to a value of one.

## TASK-VI

Synthesize the information in §2.3 to answer the following questions.

1. For the $200 \times 200$ cutout image, please run the source extraction and generate a mask for object number TWO such that everything else is set to zero. Show a two panel figure using Python, where the left panel matches Figure 4, and the right panel should show the product of the mask and the image data.

2. Compute the flux and apparent magnitude for object number TWO using the segmentation map.

3. How does this flux and magnitude compare to the aperture-based values? Justify your answer.