Model Fitting for a Toy Universe
– Project for CTA200H –

Jennifer Laing
jen.laing@mail.utoronto.ca

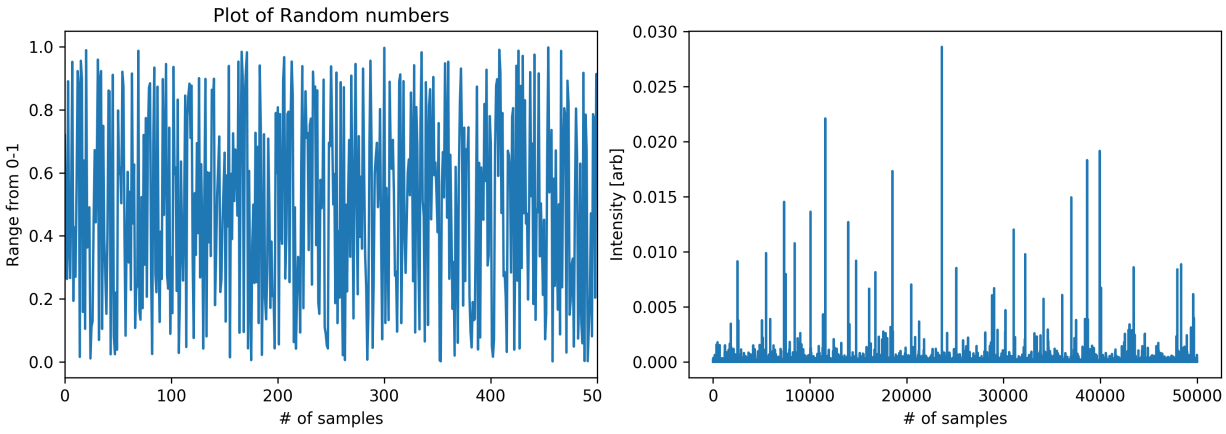
May 18, 2018

Introduction

It is common practice, in astronomical research, to generate simulations with known parameters, which can be compared to new data being studied. It's a useful tool which can help make sense of the data. In this project we are generating a 1 degree square piece of sky, introducing randomly generated flux densities, populating the image area with these flux densities at random positions, and adding Gaussian noise. The flux densities are generated according to a power law with a slope of -2.25. Python's random number generators are used throughout to generate the flux densities, the coordinate positions and the random Gaussian noise.

1 Generating Flux Densities

In order to create random flux densities, first, 50000 random numbers were uniformly generated between 0 and 1 using NumPy's random.rand function. These numbers are given the variable α. A sample from this list is plotted in Figure 1a to confirm the value and randomness of the numbers.



(a) (b)

Figure 1: Figure (a) shows a sample of the 50000 random numbers. Figure (b) shows all 50000 flux densities plotted against intensity after having the function S(α) applied.

Using the functional form of the count distribution given, $dN/dS = f(S)$, the following transformation was done to generate $S = S(\alpha)$. We started with

$$\frac{dN}{dS} = \frac{dN}{d\alpha}\frac{d\alpha}{dS} = \frac{d\alpha}{dS}, \tag{1}$$

and

$$\alpha(S) = \int f(S)ds. \tag{2}$$

Evaluating the integral with $f(S) = S^{-2.25}$

$$\alpha(S) = \int S^{-2.25}ds, \tag{3}$$

$$\alpha(S) = -0.8S^{-1.25} + C. \tag{4}$$

Making C=0 and then, inverting this, we solved for α and got a function $S(\alpha)$:

$$S(\alpha) = -1.25\alpha^{-0.8}. \tag{5}$$

The uniform values of α that were previously generated were then put through this function in order to distort the uniformity of the random numbers according to this function. The brightest sources ($S > 0.1Jy$) were discarded. A plot was made of the intensity values per sample and is shown in Figure 1b above.

A log-log histogram was generated and is shown in Figure 2. Visually, the slope appears to match the required -2.25 and a first order polynomial fit was done to confirm this using NumPy's polyfit routine. The fitted curve is shown in the plot. The slope generated by the fitting routine was -2.22.

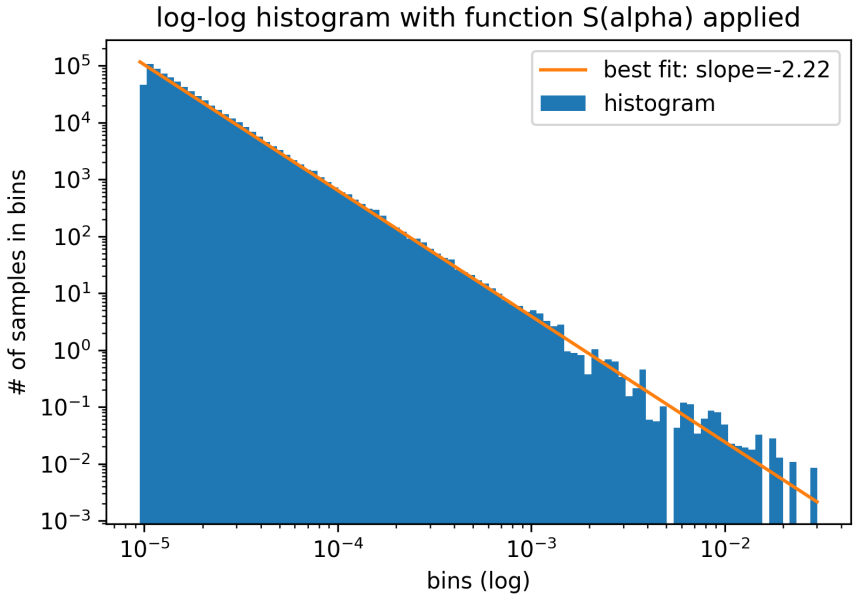


Figure 2: The log-log histogram with first order polynomial fit.

2 Create image and assign positions

In order to create the 2D image which will become the simulated image of sky, first the size of the image needed to be decided. The piece of sky is 1 degree square or 3600 arcseconds square. The 15 arcsecond beam must be properly sampled and not oversampled. It's too computationally heavy to have a one to one ratio of one arcsecond per pixel and anything close to 15 arcseconds per pixel will be oversampled. It was decided to use something in between those while keeping the pixel size as small as possible. A pixel size of 4 arcseconds was chosen, so the image will be 900 pixels square.

A 2D zeros array was created at 900 X 900 pixels. Using NymPy's random integer generator, random.randint, a random sample of 50000 values were pulled from a range of zero to 900. This was done twice to generate x and y coordinates for each of the sources. In order to check that these were randomly distributed a scatter plot was generated of a sample of 200 of the values and is shown in Figure 3.

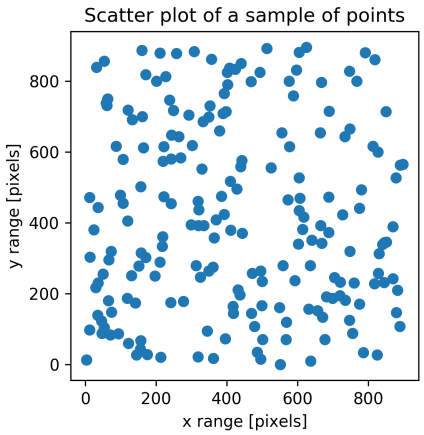
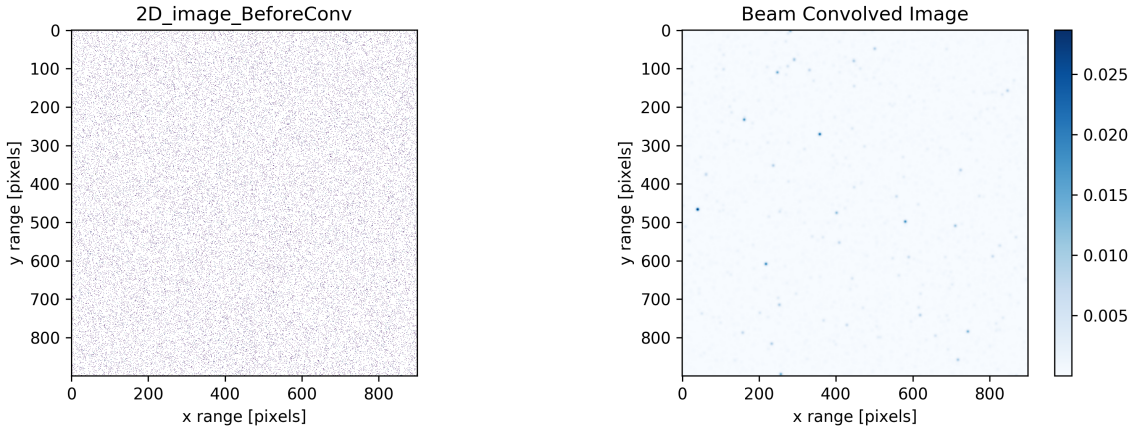


Figure 3: Scatter plot shows that the points are randomly distributed.

The 2D zeros array was then populated, assigning the random x and y coordinates to each of the flux densities generated previously. In the case where any coordinates overlapped on one pixel, a for loop was written to ensure that all sources on that pixel would be added together rather than replaced. Figures 4a and 4b show the 2D image before and after 2D convolution is done. This will be explained in the next section.



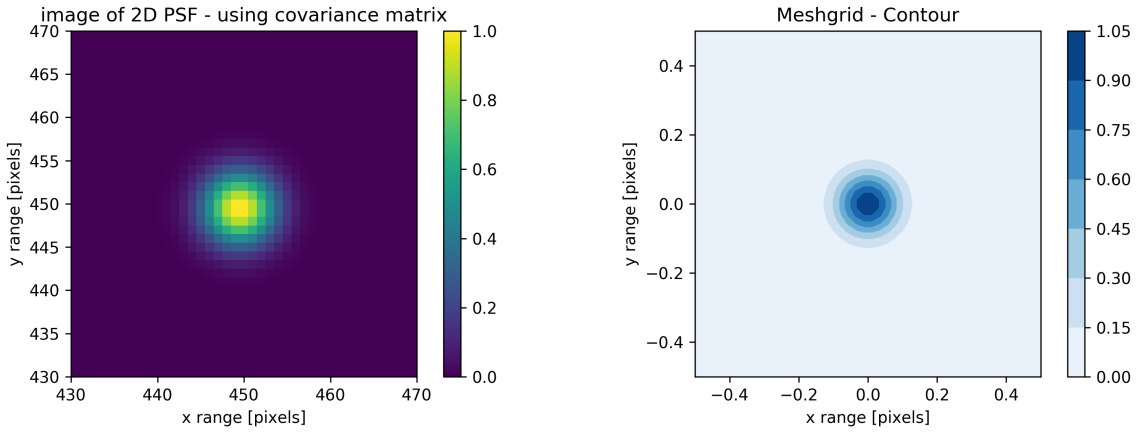
(a) (b)

Figure 4: Figure (a) shows the 2D image populated with flux densities before the convolution and Figure (b) shows the result of the 2D convolution of the beam and image, done in the next section.

3 Generate Beam and Convolve with Image

Next, the Gaussian beam with FWHM 15 arcseconds was created. This was done by first preparing a meshgrid on which to plot the resulting image. Both 1D and 2D Gaussian functions were written and/or

adapted from one of our lectures. The FWHM was defined to be 15 divided by 4 since there are 4 arcseconds in a pixel. The 2D Gaussian function was called in order to generate the PSF. Plotting an image of the PSF shows the 2D image of the Gaussian, shown in Figure 5a as an imshow image and 5b as a contour image. The peak is normalized to 1.



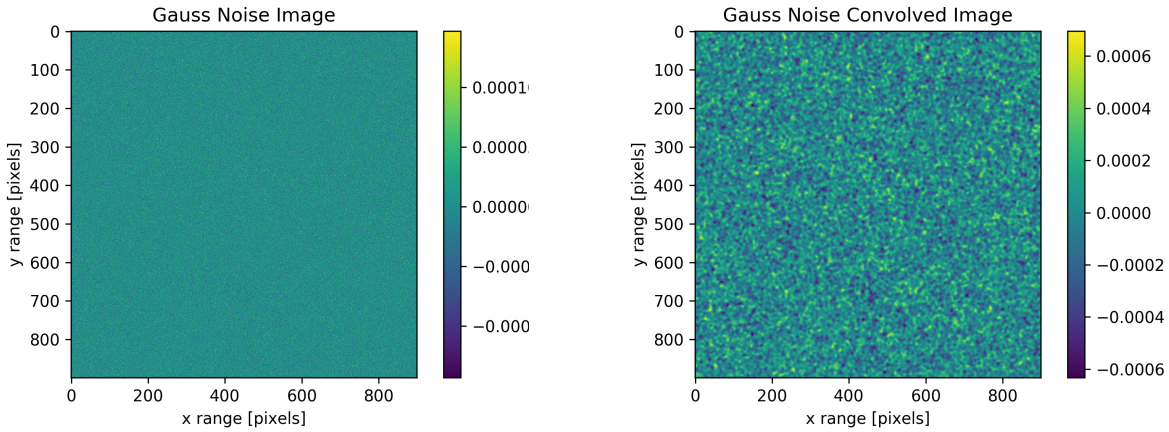
(a) (b)

Figure 5: Figure (a) is a 2D image plot of the 2D Gaussian function, and Figure (b) shows a contour image of the same.

Using Scipy's signal.fftconvolve routine the 2D image with flux positions was convolved with the 2D Gaussian beam to generate a softer more realistic image of the sources, shown above in Figure 4b.

4 Make Some Noise

A 2D array of gaussian random noise was generated using NumPy's random.normal routine. The resulting image is shown in Figure 6a. And then, using the same fft Convolution routine the Gaussian noise image was convolved with the Gaussian beam image and is shown in Figure 6b. In this case, σ is set to 30 μJy.



(a) (b)

Figure 6: Figure (a) shows the image generated for random Gaussian noise. Figure (b) shows the Gaussian noise image after convolution.

A histogram of the beam convolved Gaussian noise image was created. And using the 1D Gaussian function created earlier, a plot of the theoretical model line was plotted on top. This plot is shown in Figure 7. The histogram seems very thin in comparison with the Gaussian curve. A variety of different bin sizes were tried with similar results.

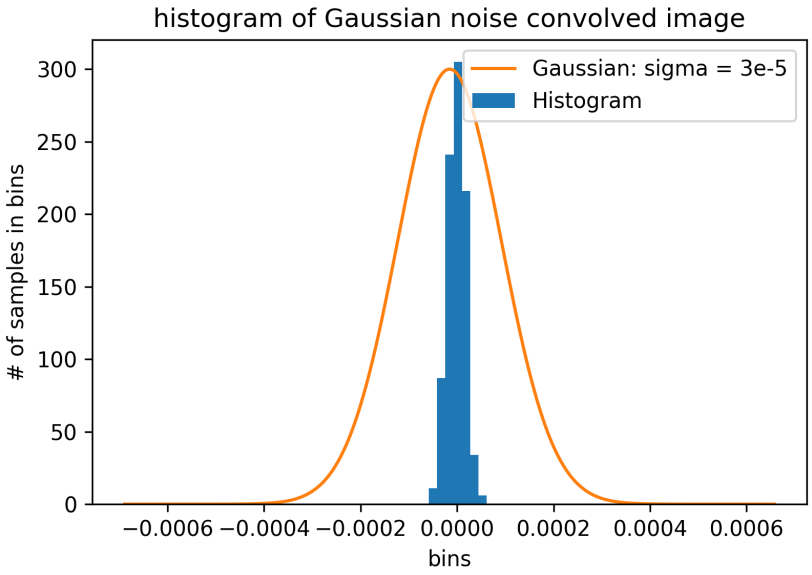


Figure 7: Histogram of the beam Convolved Gaussian noise image with theoretical Gaussian curve.

The beam Convolved source image and beam Convolved noise image were then added together to create a more realistic image since real data would contain a lot of noise. This noisy image is shown in Figure 8 on the next page. Figure 9 shows a close up of an area of this image.

The next stage will be to improve upon this image if possible and then to create a fits file. The Aegean source finding algorithm will then be used to experiment with finding sources in the image. The final step will involve counting sources and comparing input and output source counts.

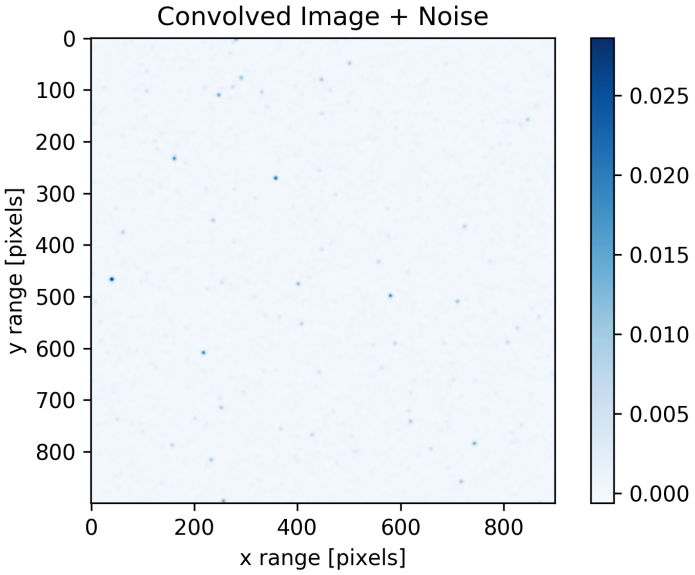


Figure 8: Noisy image.

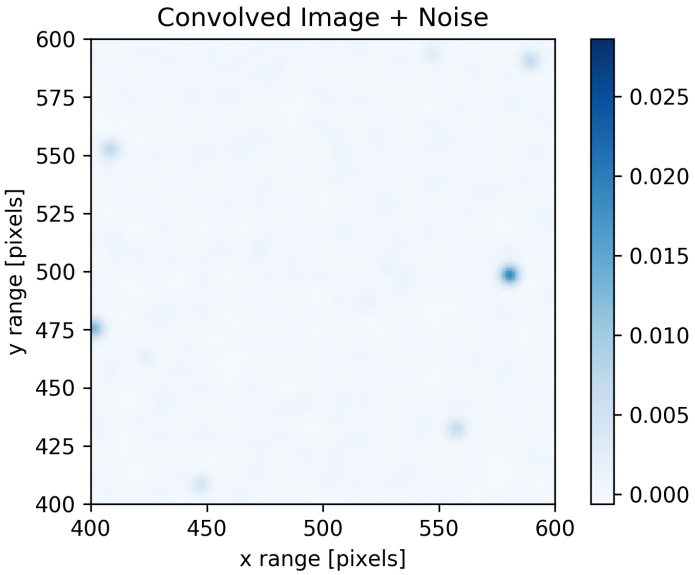


Figure 9: Close up on noisy image.