Obs & Stats HW 3

Ryan Hofmann

15 February 2017

Uncertainties in Electron Counts on a CCD detector

1. Electrons accumulate according to the Poisson distribution:

$$f(k) = \frac{\lambda^k}{k!} e^{-\lambda} \tag{1}$$

2. Given that

$$V[k] \equiv E[(k-\lambda)^2] = E[k^2] - E[k]^2 = E[k^2] - \lambda^2 \tag{2}$$

and evaluating $E[k^2]$ as

$$\begin{aligned} E[k^2] &= \sum_{k\geq 0} k^2 \frac{1}{k!} \lambda^k e^{-\lambda} \\ &= \lambda e^{-\lambda} \sum_{k\geq 1} \frac{k}{(k-1)!} \lambda^{k-1} \\ &= \lambda e^{-\lambda} \left(\sum_{k\geq 1} \frac{k-1}{(k-1)!} \lambda^{k-1} + \sum_{k\geq 1} \frac{1}{(k-1)!} \lambda^{k-1} \right) \\ &= \lambda e^{-\lambda} \left(\lambda \sum_{k\geq 2} \frac{k-2}{(k-2)!} \lambda^{k-2} + \sum_{k\geq 1} \frac{1}{(k-1)!} \lambda^{k-1} \right) \\ &= \lambda e^{-\lambda} \left(\lambda \sum_{i\geq 0} \frac{i}{i!} \lambda^{i} + \sum_{j\geq 0} \frac{1}{j!} \lambda^{j} \right) \\ &= \lambda e^{-\lambda} (\lambda + 1) e^{\lambda} \\ &= \lambda(\lambda + 1) \end{aligned} \tag{3}$$

it follows that

$$V[k] = E[k^2] - \lambda^2 = \lambda^2 + \lambda - \lambda^2 = \lambda \tag{4}$$

3. The standard deviation is $\sigma \equiv \sqrt{V[k]} = \sqrt{\lambda}$ electrons.

4. The mean is $\mu \equiv E[k/G] = \frac{1}{G} E[k] = \frac{\lambda}{G}$ DN.

5. The standard deviation is $\sigma \equiv \sqrt{V[k/G]} = \sqrt{\frac{V[k]}{G^2}} = \frac{\sqrt{\lambda}}{G}$ DN.

6. In a Poisson distribution, the mean and variance are equal, and the standard deviation is the square root of the variance, i.e. $\mu = \lambda = \sigma^2$. For the DU distribution, this is no longer the case, as $\mu = G\lambda$. Therefore, to estimate the noise in a CCD image, it is essential to first convert from DU to electrons.

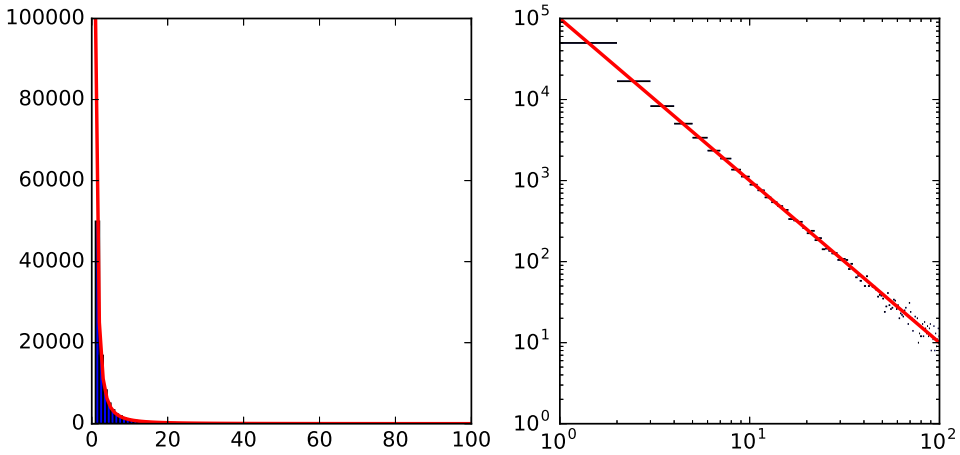

Figure 1: Power law distribution, generated from an analytic $x(\alpha)$

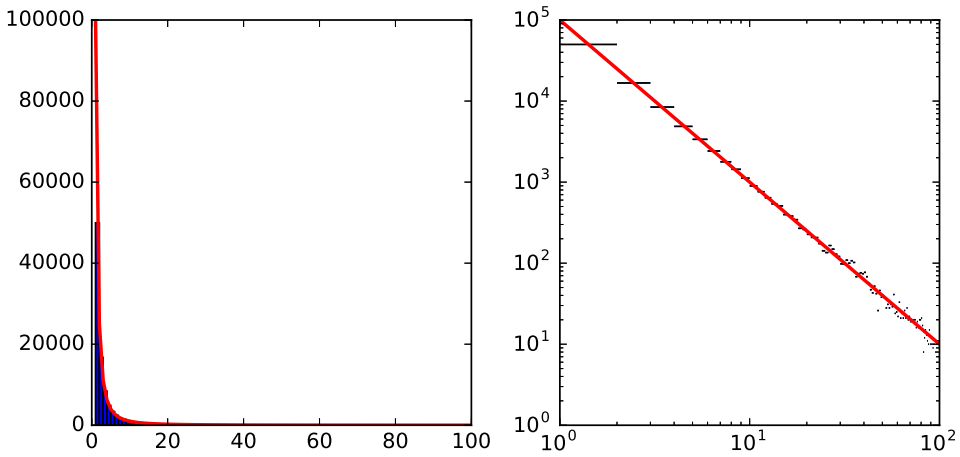

Figure 2: Power law distribution, generated from a numerical solution to $x(\alpha)$

Drawing random numbers from analytic distributions

1. Given $\alpha(x) = \int^x f(x)dx$ with $f(x) = x^{-p}$, integrate and solve for x:

$$\begin{aligned} \alpha(x) &= \int_1^x x^{-p}dx \\ &= \frac{1}{-p+1}(x^{-p+1} - 1) \\ (-p+1)\alpha &= x^{-p+1} - 1 \\ x(\alpha) &= [(-p+1)\alpha + 1]^{\frac{1}{-p+1}} \end{aligned} \tag{5}$$

2. See Figure 1. The excellent agreement between the generated and analytic distributions can be most clearly seen in the log-log plot.

3. The numeric solution, shown in Figure 2, is a very close match to the analytic solution in Figure 1.

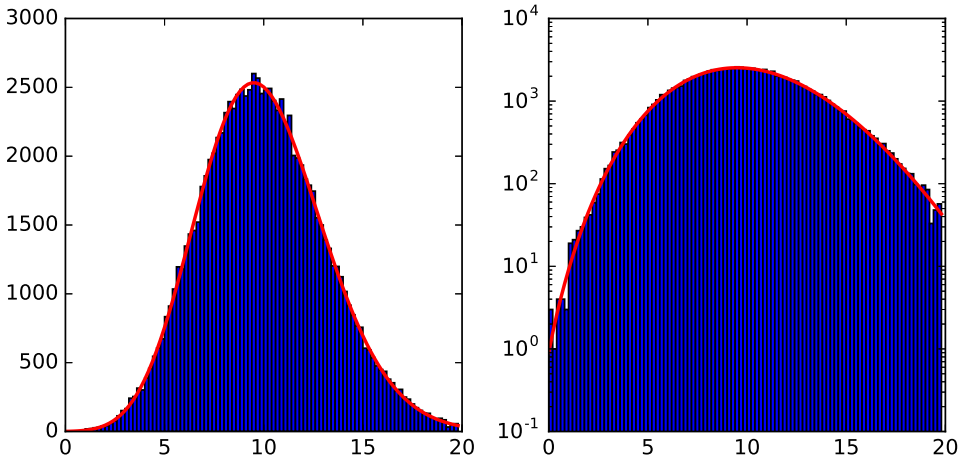

Figure 3: Poisson distribution, generated using a custom numeric function. The analytic curve has been scaled down by a factor of 5 to account for the bin size.

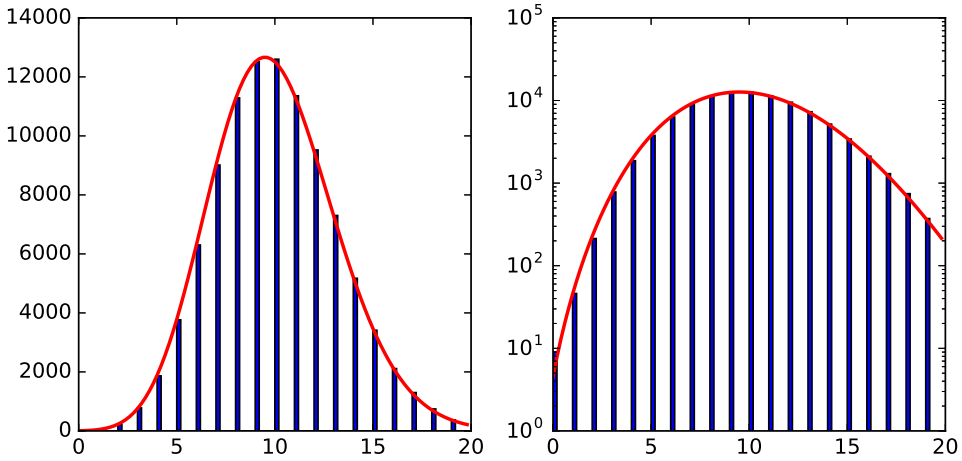

Figure 4: Poisson distribution, generated using np.random.poisson()

Monte Carlo Generation of a Random Number from a Poisson Distribution

1. See Figure 3. The agreement between the numeric and expected distributions is excellent.

2. See Figure 4. Whereas my Poisson generator outputs a float, np.random.poisson() only produces integers. Thus, when sub-unit bin sizes are used, the float histogram is scaled down, while the integer histogram's filled bins get narrower but remain at the same height. The two can be made to appear the same by using unit-width bins for both.

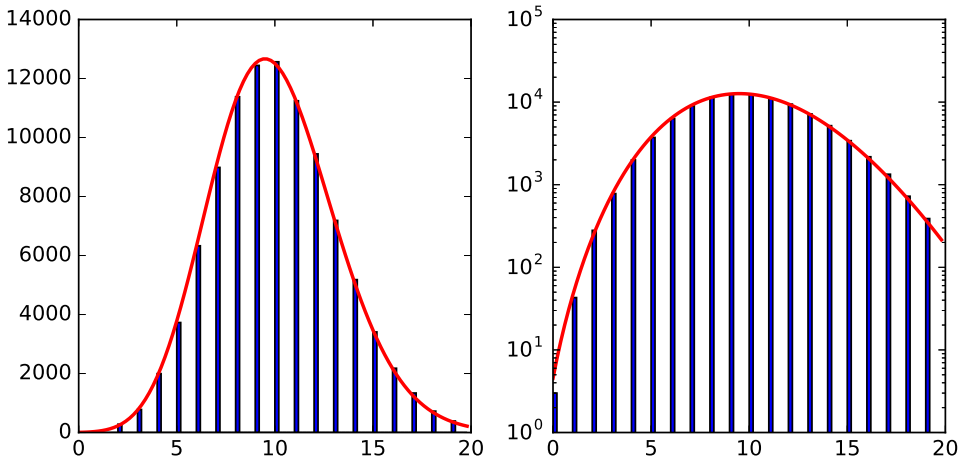

Figure 5: Poisson distribution from custom function, rounded to nearest integer

3. My Poisson generator itself returns the interpolated function $x(\alpha)$, which then acts on an array of uniform random numbers; I was unable to figure out a "nice" way of having the generator actually produce a random number without having to recalculate the interpolation function every time it was called, which would have been prohibitively computationally expensive. Therefore, instead of modifying the function itself, I can modify the line where the function is called by using numpy.rint() to round the transformed random number to the nearest integer. This reproduces the appearance of Figure 4 almost exactly, as can be seen in Figure 5.