Antony Sikorski
Phys 164
Professor Konopacky
10/15/2020

Physics 164 Lab #1: Photon Counting and Detectors

Author: Antony Sikorski, Lucas Scheiblich, Yizhang Liu
Contact: asikorsk@ucsd.edu, lscheibl@ucsd.edu, yil103@ucsd.edu
11, March, 2020

Abstract

Modern astronomy relies on using Charge-Coupled Device (CCD) integrated circuits that use the photoelectric effect to observe light at ultraviolet, infrared, and visible wavelengths. Current detectors are made of semiconductor materials which convert incoming photons into electrons, which are transformed into digital information that computers use to build an image of what a telescope is observing. During this experiment, the Nickel 1-m telescope at Lick Observatory sent information to the installed CCD to record multiple different images with varying exposure times in order to assess the accuracy and shortcomings of a modern detector in regards to photon counting. The images of varying exposure, along with blank bias frames, were analyzed to determine the relationship between exposure time and image accuracy, along with identifying systematic sources of noise which could produce disruptions in the data. The analysis included thorough statistical modeling of the means and standard deviation of the photon counts, along with comparison to theoretical Poisson and Gaussian distributions. Methods for identifying camera gain and read noise were developed in order to ensure higher accuracy in further experiments. A positive correlation was observed between the exposure times and the accuracy of the images, as the mean and standard deviation of the data slowly converged to be similar to theoretical probability distributions, as predicted by the central limit theorem.

Section 1: Introduction

The photoelectric effect is described as an interaction between photons and bound electrons. When a photon above a certain threshold frequency comes in contact with a surface, it will increase the kinetic energy of an electron, which causes the electron to become excited. An electron that is excited in this manner is called a photoelectron, and it can now eject from its energy level to travel freely, producing the desired effect.

A charge-coupled device (CCD) detector is a circuit composed of a two dimensional array of pixels that have been etched onto a surface of a semiconductor material such as silicon. Semiconductors are materials with a relatively small gap in between their valence and conduction bands, meaning they are highly sensitive to this effect, and electrons can enter the conduction band quite easily when excited by a photon. The light sensitive pixels utilize the photoelectric effect to record data by counting the number of incident photons on the surface. This information travels down a column of pixels to accumulate at a charge amplifier, which sends it to an analog to digital converter, which produces measurable data that a computer can then use to produce an image.

The longer the exposure time, the more photons are collected by these pixels, and theoretically the accuracy of photon counts across all of the pixels on the detector improves. Unfortunately, with high exposure time comes the price of multiple different sources of noise. Read noise and gain are primary issues, with read noise being produced when electrons are subject to the analog to digital conversion, and gain being a multiplicative property of detectors that is unavoidable. There are also dark currents, which are produced by thermal effects and warp the data. Hot pixels and photon noise are also consequences of higher exposure; more photons creates a higher chance for random error, and also increases the probability of overheating a pixel.

In this experiment, pictures were taken with multiple exposures in order to study the accuracy of the photon counting properties of the detectors. The sources of error that were primarily studied were read noise and gain, which were calculated to improve the accuracy of later experiments which will deal with the exact same error. Hot pixels and other bad pixels were observed by plotting the actual files in order to isolate clean areas of the detector. Bias frames with zero exposure were also taken into account, because it is likely for there to still be some form of information recorded without ever exposing the detector. With all of these sources of error being removed, the theory that a higher exposure will produce a more accurate photon count was systematically tested, and proven to be correct.

Section 2: Observations and Data

The equipment used for this lab consisted of the Nickel 1 Meter telescope, located in the dome at the north end of the Lick Observatory in California. As stated in the name, the length of the telescope is 1 meter, and it is primarily used for imaging purposes. Behind the primary mirror of the telescope is the Direct Imaging Camera, which contains a 2048 x 2048 pixel CCD array. It is critical to maintain the temperature of the array to avoid error caused by thermal effects, so it is cooled to a degree of 77 Kelvin.

The telescope is controlled via graphical user interfaces that allow the user to control the dome, lights, camera, and other properties of the telescope. All control and operations were conducted by Professor Konopacky, with the rest of the Physics 164 class observing the process.

The process was as follows: First, 4 images with 0 seconds of exposure were taken to be used as Bias frames, meaning they would be subtracted from the data to account for the signal that was present despite the lack of light. After this, 4 sequences of images with varying exposure times were accidentally taken with the shutter closed. This data was disregarded. After re-opening the shutter of the camera, 3 images of each exposure time were taken, with exposure times of 3, 6, 12, 48, and 384 seconds. Additional exposure time data from the 2019 experiment was used to supplement the 24, 96, and 92 second exposure times. The images taken were instantly processed by the CCD and turned into FITS (Flexible Image Transport System) files. This is the standard file type for recording astronomical image information, and the data is stored in Binary form. The Python programming language was used via the UCSD JupyterHub interface in order to analyze the properties of the imported FITS files.

Section 3: Data reduction and Methods

In order to analyze the data in the most efficient way possible, we chose to use exposure times of 3, 6, 12, 24, 96, and 192 seconds. Three files of each exposure time were used in order to lower the variability and get a better understanding of the photon counting dynamics at each exposure time. Next, the bias data was subtracted from each individual frame of each exposure time in order to reduce error and have the highest accuracy possible. The FITS files were then plotted in order to estimate additional errors, such as overscan regions, dark currents, and hot pixels, that could not be easily noticed without visual analysis. An example of hot pixels and overscan regions can be viewed in **Figure 1** below.

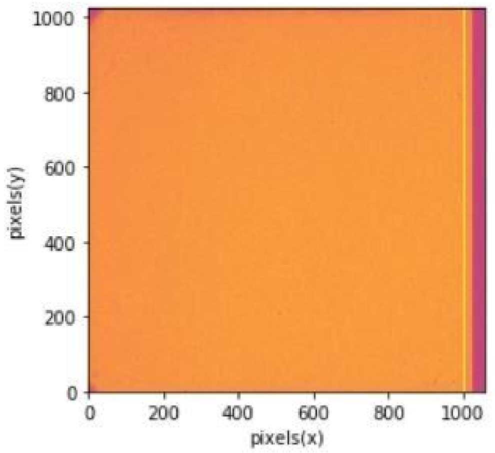


Figure 1: Image of FITS file img6_2. This particular example is the second file in the 6 second exposure time sequence.

It can be clearly seen that there is an overscan region past 1024 pixels on the x axis. The data was cropped to be 1024 x 1024 pixels to avoid having this ruin the data. Hot pixels can be observed as irregularly yellow pixels in the graph, although they are more difficult to see.

Improving pixel accuracy was done via averaging the pixel counts together and creating histograms of their frequency **(Figures 2-8)**. Variance was then analyzed by looking at the means of each sequence of images with the same exposure time, and then calculating the standard deviation of these means. To prove that the accuracy increases with exposure time, the data was compared to the theoretical Poisson and Gaussian distributions. Finally, to improve further accuracy, the read noise and gain factor were calculated via a least squares regression model of the six clearest FITS files from each different exposure sequence.

Section 4: Calculations and Modeling

First, each FITS file of all varying exposures was turned into a histogram to observe the peaks and the frequencies of different counts (ADU).

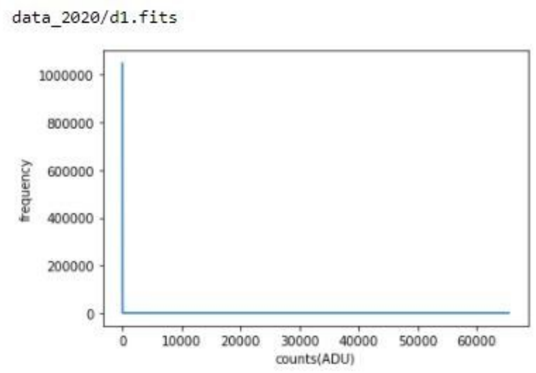


Figure 2: Exposure Time: 0 File used: d1.fits

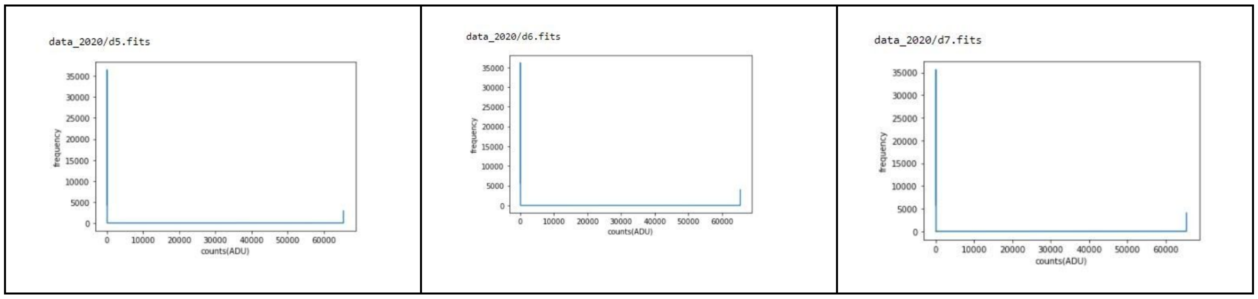


Figure 3: Exposure Time: 3 Files used: d5.fits, d6.fits, d7.fits:

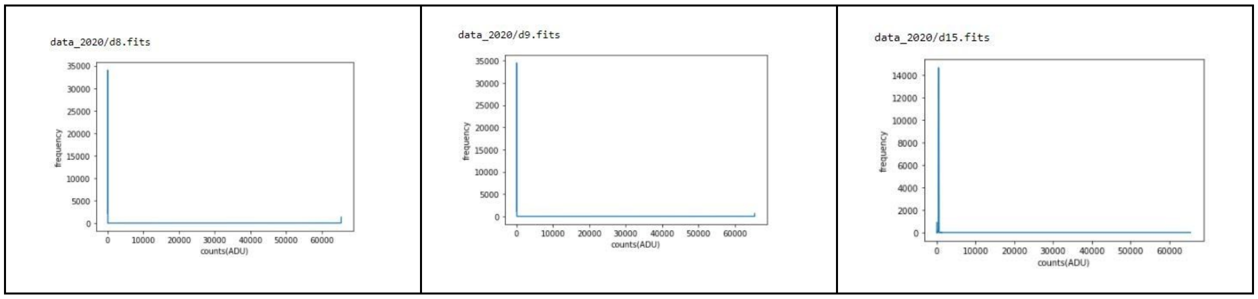


Figure 4: Exposure Time: 6 Files used: d8.fits, d9.fits, d15.fits:

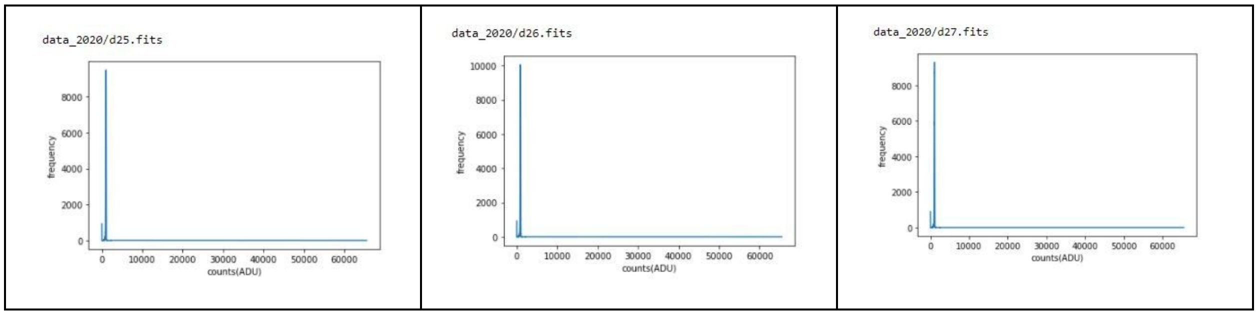


Figure 5: Exposure Time: 12 Files used: d25.fits, d26.fits, d27.fits:

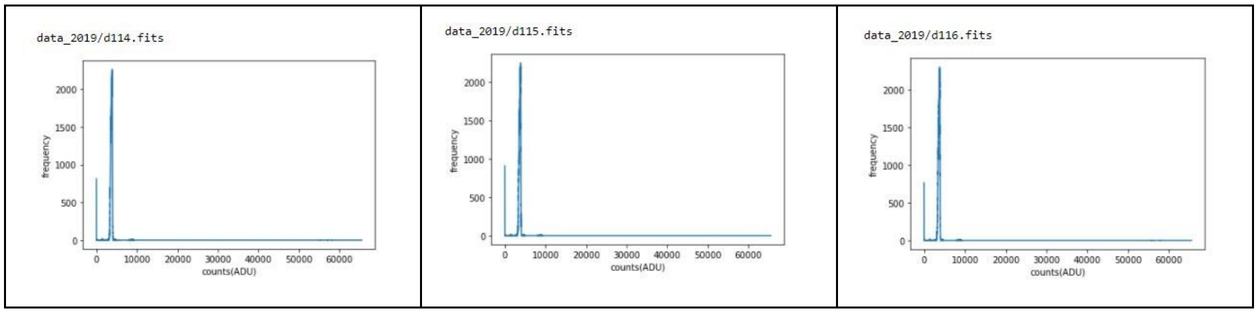
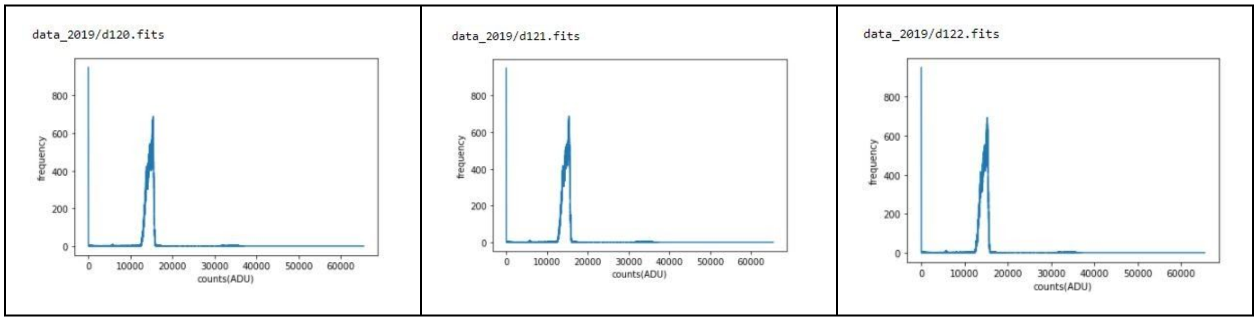


Figure 6: Exposure Time: 24 Files used: d114.fits, d115.fits, d116.fits:



Figures 7: Exposure Time: 96 Files used: d120.fits, d121.fits, d122.fits:

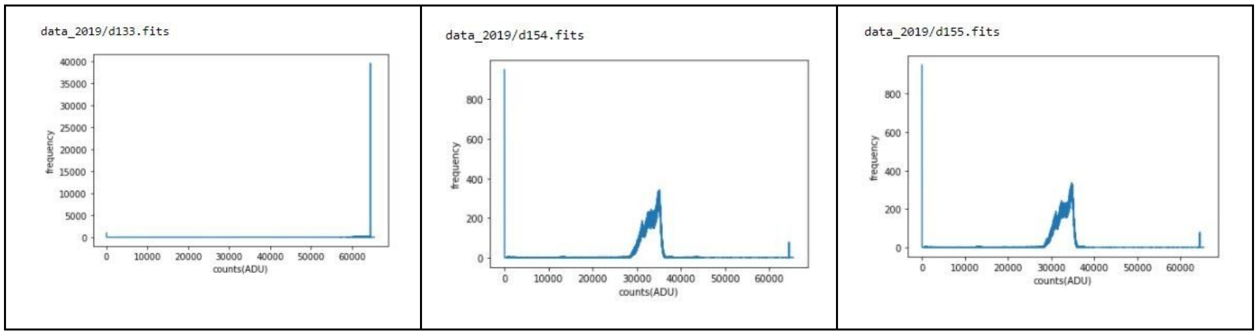


Figure 8: Exposure Time: 192 Files used: d133.fits, d154.fits, d155.fits

:

After creating the histograms, we used the equations for mean and standard deviation (**Equations 1,2)** to compute these values for each file's photon counts, and then plotted them as a function of exposure time to investigate the correlation between the size of the mean and standard deviation relative to the exposure time (**Figure 9**). Each value was individually recorded in **Table 1**. We neglected to use the bias FITS file due to the fact that it had already been subtracted from all of the other files.

Equation 1: $\mu = 1/N \sum^{N}_{0} x$

Equation 2: $\sigma^2 = 1/N \sum^{N}_{0} (x-\mu)^2$

μ = mean **N** = length of list **x** = element of list **σ**=standard deviation

Table 1: Means and Standard Deviations of each FITS file

Exposure Time	Mean	Standard Deviation
3	1077.4190877262415	1940.6690275205256
3	1076.4787921311338	1954.8515288195908
6	1082.407562255084	1957.509067731295
6	1084.1948501122954	1939.389250573559
6	1544.3487909490675	1947.945769254166
12	2013.2104889839745	1955.9323805763659
12	1945.8187542538371	1953.0572327432822
12	2045.1102405893603	1951.3437002182973
24	4643.603842995799	1976.1443281647157

24	4637.245868105677	2001.4029678757167
24	4598.333346280154	1963.6161552455576
96	15170.479819557986	3188.047022297359
96	15222.524936560389	3194.0151924568613
96	15097.788007332505	3174.5594496613526
192	62349.254910237854	11244.388328037578
192	33259.03258630045	6295.652319453716
192	33019.429342560004	6250.182727290769

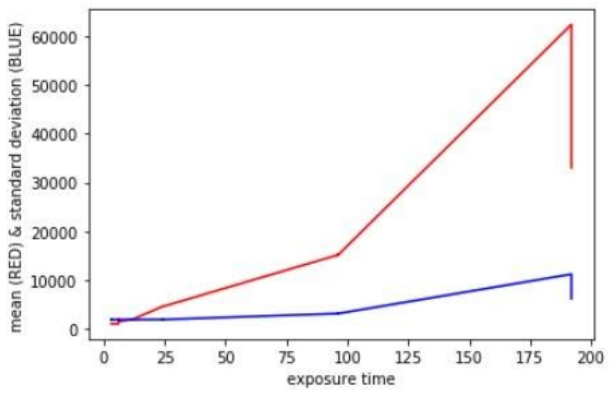


Figure 9: Plot of the mean and standard deviation vs exposure time

The most clear file of each exposure time (3,6,12,24,96,192) was then compared to both the Poisson and Gaussian distributions to compare the shape of the two distributions. The equations for these theoretical probability distributions are shown below as **Equations 3,4**. The plots of the distributions are shown below as **Figure 10**.

Equation 3: $(;) = {}^{x-}/! = {}^{x-}/((2\pi x)^{1/2}(x/e)^{x})$

P = Poisson Distribution = mean **σ**=standard deviation

Equation 4: $G(;,) = 1/(2)^{1/2}\ {}^{-1/2}\ ((-)/)^{2}$

G= Gaussian Distribution = mean **σ**=standard deviation

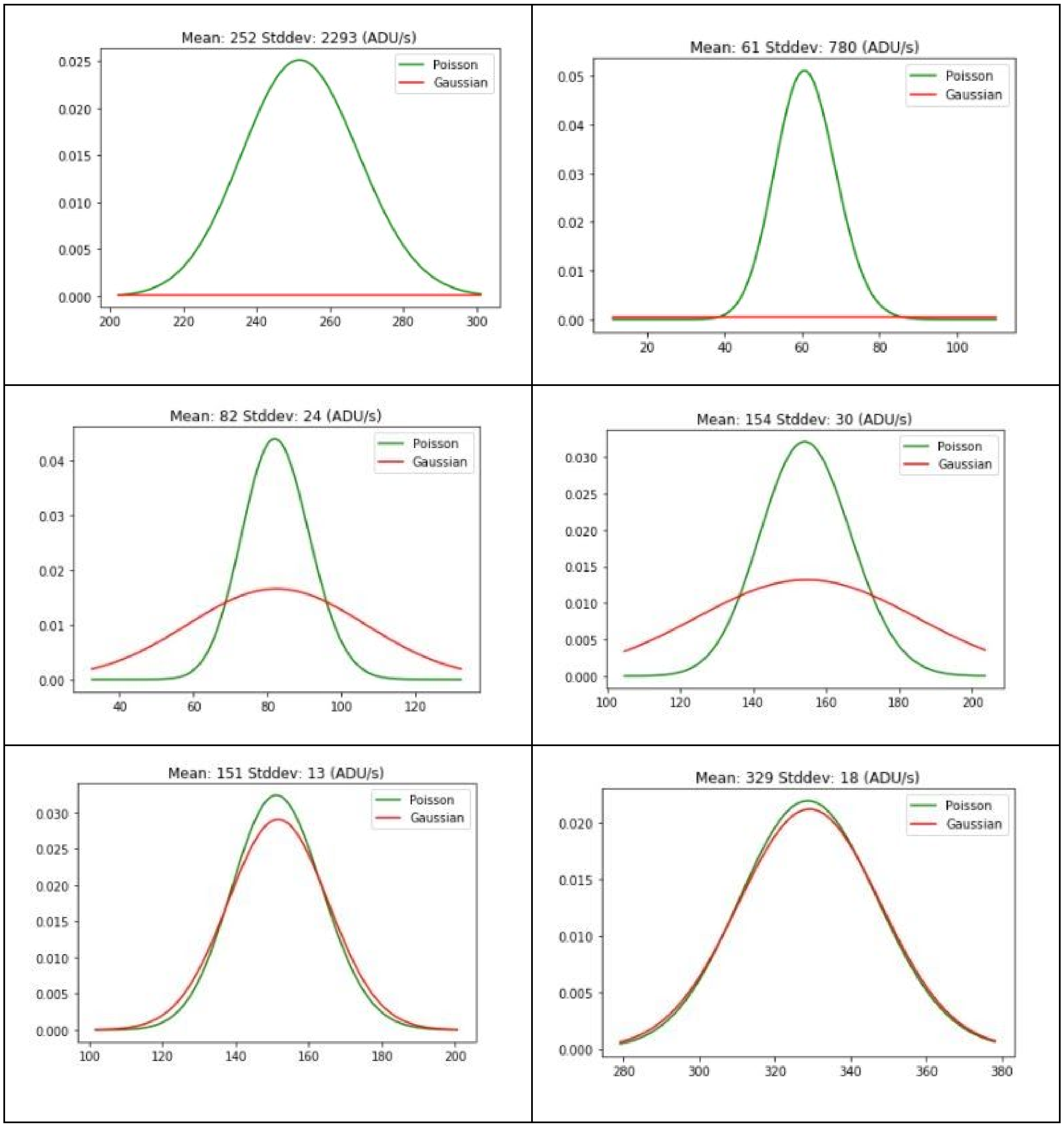


Figure 10: Gaussian and Poisson plots of the 3, 6, 12, 24, 96, and 192 second exposure times.

Next, the mean and standard deviation of each subsequence of each exposure time was taken to view the variability in between the same exposure times. These were again calculated using **Equations 1,2**. They are shown below in **Table 2**.

Table 2: Mean and standard deviation of the means of each subsequence of exposure times

Exposure Time Group	Mean of Means	Standard Deviation of Means
3	1076.691232698928	0.529325357007094
6	1236.9837344388156	217.34114056345288
12	2001.3798279423906	41.389799227909464
24	4626.394352460543	20.01118040942353
96	15163.59758781696	51.155639725602114
192	42875.905613032766	13770.084775019146

Finally, the camera gain factor and read noise were calculated using **Equation 5**, and by making a plot of the variance as a function of the mean of each pixel within the fits files. The plot is first shown on a logarithmic scale for clarity **(Figure 11)**, and then reverted back in order to be able to find the linear fit using the least squares method **(Figure 12)**.

Equation 5: $\sigma_{ADU}{}^2 = ge/C(ADU - ADU_o) + \sigma_o{}^2$

$\sigma_{ADU}{}^2$ = variance $\sigma_o{}^2$ = Read noise **ADU** = mean of images/dark images ***ge/C***= Gain

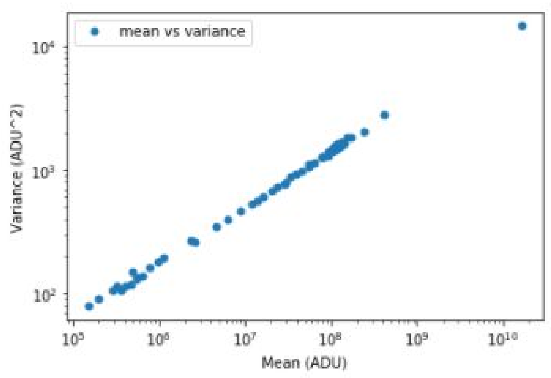


Figure 11

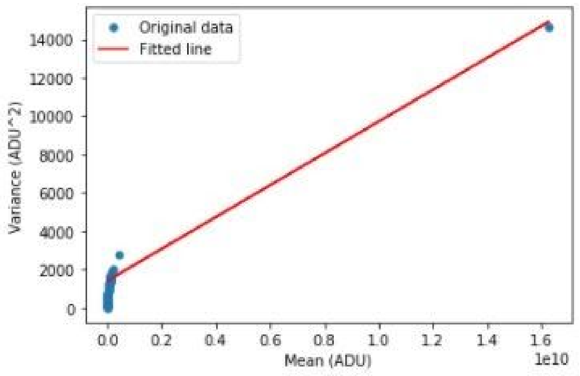


Figure 12

The gain was found to be 2.86754e-08 and the error of the read noise was 14.96880.

Antony Sikorski
Phys 164
Professor Konopacky
10/15/2020

Section 5: Discussion

By the central limit theorem, for a population of data with a given mean and standard deviation, the probability distribution will approach a normal/Gaussian distribution. A very good approximation for a normal distribution is a Poisson distribution with a high number of counts, which is exactly why we modeled our data using both distributions. This helped to approximate the relationship between the accuracy of the photon count and the exposure time of the image. Our theory was proven to be true, as shown by Figure **10**, where the two distributions converged as the exposure time of the file increased. Looking back at **Figure 8** it is also evidenced that the actual spread of pixel counts approaches these two distributions just by observing the shape of the actual data. It is finally confirmed when plotting the means and standard deviation of all of the files against the exposure time in **Figure 9**, where the mean continues to rise while the standard deviation becomes smaller and smaller by comparison. It appears to be true that as the exposure time is increased on an image processed by a CCD, the actual photon count data becomes significantly more accurate.

If this experiment were to be repeated again, there are several areas of improvement that I would suggest. To begin, a better spread of exposure times would lessen the drastic changes shown by **Figures 2-8**, in which the shape of the photon count histogram changes drastically over the course of our selected exposure times. Additionally, data was used from the same experiment that was conducted last year, which is not preferable, because there are many conditions that could be different (weather, temperature, etc) if the data is not taken at the same time. Finally, in actuality, very little noise was removed from the data in order to maximize the accuracy of this experiment. While the overscan region was removed by cropping the images, and both the read noise, gain, and error were calculated, we still did not account for several anomalies such as actually removing dark pixels, and accounting for linearity and saturation. We may have noticed regions with bad pixels, which allowed us to better understand certain anomalies, but the physical removal of these via Python would allow for a much closer to ideal data set. In the end, our gain was found to be 2.86754e-08, and our error of read noise was found to be roughly 15% (shown after **Figures 11&12**) which is a high enough statistic to cause some concern. To be entirely fair, with the time given and the data sets provided, the different methods for examining the data were incredibly efficient in helping better understand the relationship between the accuracy and exposure, and the limitations of the current method that astrophysicists use to detect light.

Antony Sikorski
Phys 164
Professor Konopacky
10/15/2020

Section 6: Appendix

All of the in house functions and figures were created by myself and my lab group in a collaborative effort. Our lab group consists of myself, Antony Sikorski, and my lab partners Lucas Scheiblich and Yizhang Liu. No particular function was dominated by one of the members of the lab group, because each one was thoroughly verified and improved by the remaining group members. Code was also sampled and used from most of the Discussions and Coding Exercises, which was permitted by Caleb Cohan, the primary creator.

In House Functions (All written in Python):

Mean

This function sums each element divided by the length of the actual array in order to compute the mean.

```python
def inhousemean(x):
    avg = 0
    for i in x:
        avg += i/len(x)
    return avg
```

Standard Deviation

This function performs the sum of squares and then inserts it into the equation for standard deviation.

```python
def inhousestd(x):
    avg = inhousemean(x)
    sd = 0
    for i in x:
        sd += ((i - avg)**2)
    std = (sd/(len(x)))**(1/2)
    return std
```

Histogram Generator

This function flattens the data, subtracts the bias, and then runs a loop to generate a histogram for each function.

```python
def histo(fit):#Input fit file name, ouput histogram.
    arr=np.array(fits.getdata(fit)) #convert the fit name to usable data
    arr=arr[:,0:1024] #truncate off the overscan region
    x=arr.flatten()-bias #flatten the data and subtract the bias data
    hmin=0
    hmax=arr.max()
    hist = []
    hr=np.arange(hmin,hmax+1)
    for i in hr:
        c = len(np.where(x==i)[0])
        hist.append(c)
    plt.plot(hr,hist)
    plt.xlabel('counts(ADU)')
    plt.ylabel('frequency')
    plt.show()

    #Wrote a function to loop through all of my histograms
for i in imgarr:
    print(i)
    histo(i)
```

Mean/SD Plot

This function combines all of the means and standard deviations and creates a plot of them against the exposure time.

```
TIME = [3,3,3,6,6,6,12,12,12,24,24,24,96,96,96,192,192,192]

inhousemeans =
[inhousemean(x1),inhousemean(x2),inhousemean(x3),inhousemean(x4),inhousemean(x5),inhousemean(x6),inhousemean(x7),
inhousemean(x8),inhousemean(x9),inhousemean(x10),inhousemean(x11),inhousemean(x12),inhousemean(x13),inhousemean(x14),
inhousemean(x15),inhousemean(x16),inhousemean(x17),inhousemean(x18)]

inhousestds =
[inhousestd(x1),inhousestd(x2),inhousestd(x3),inhousestd(x4),inhousestd(x5),inhousestd(x6),inhousestd(x7),
inhousestd(x8),inhousestd(x9),inhousestd(x10),inhousestd(x11),inhousestd(x12),inhousestd(x13),inhousestd(x14),
inhousestd(x15),inhousestd(x16),inhousestd(x17),inhousestd(x18)]

plt.plot(TIME,inhousemeans,'r',TIME,inhousestds,'b')
plt.xlabel('exposure time')
plt.ylabel('mean (RED) & standard deviation (BLUE)')
plt.show()
```

Poisson & Gaussian Distributions

This function plots the Poisson and Gaussian distributions for each mean and standard deviation given from the clearest FITS file of each exposure time.

```
def dists(means,stds): #generate plots
    for i in range(len(means)):
        plt.figure()
        m = means[i]
        s = stds[i]
        r = np.arange(m-50,m+50) #the distributions were off-center so I chose to vary r based on the mean.
        #Poisson
        pDist= 1 / np.sqrt(2 * np.pi * r) * np.power (m * np.exp(1.0) / r, r) * np.exp(-m)
        plt.plot(r, pDist, c = 'g', label = 'Poisson')
        #Gaussian
        gDist= 1 / (s * np.sqrt(2*np.pi)) * np.exp(-1/2 * np.power((r-m) / s, 2))
        plt.plot(r,gDist,c = 'r', label = 'Gaussian')
        plt.title('Mean: %i Stddev: %i (ADU/s)'%(m,s))
        plt.legend()

dists(xbar,sdevs)
```

Camera Gain & Read Noise

This function effectively takes the mean of each pixel in the array, computes the variance of each, and then plots the data to find the linear fit using the least squares. The final function was given in discussion 3 which allows us to compute the read noise and gain.

```
def pixelmean(arr): #takes the mean of pixels in the array
    N=len(arr)
    s=[]
    i=0
    while i<1024:
        a=[arr[0][i],arr[1][i],arr[2][i],arr[3][i],arr[4][i]]
        s.append(inhousemean(a))
        i+=1
    return np.array(s)

def pixelvar(arr):#takes the variance of pixels in the array.
    N=len(arr)
    s=[]
    i=0
    while i<1024:
        a=[arr[0][i],arr[1][i],arr[2][i],arr[3][i],arr[4][i]]
        s.append((inhousestd(a)*5)**2)
        i+=1
    return np.array(s)

x=pvar
y=pmean
A = np.vstack([x, np.ones(len(x))]).T

m, c = np.linalg.lstsq(A, y, rcond=None)[0]
print(m,c)

%matplotlib inline
```

```
# Plot original data
plt.plot(x, y, 'o', label='Original data', markersize=5)
# Plot fit
plt.plot(x, m*x + c, 'r', label='Fitted line')
plt.ylabel('Variance (ADU^2)')
plt.xlabel('Mean (ADU)')
plt.legend()

def e_gain(sy,x): #Input stddev of y axis and the x values as an array.
    N=len(x)
    d=N*np.sum(x**2)-np.sum(x)**2
    return [sy*np.sqrt(N/d),sy*np.sqrt(np.sum(x**2)/d)] #output gain and read noise error
respectively
error=e_gain(inhousestd(y),x)
print('The error of the gain is ',error[0])
print('The error of the read noise is ',error[1])
```