Lab 1: Basic Photon Statistics with Python

Victoria Spada

October 3, 2022

1 Reproducing The Poisson Distribution

Figure 1 was made to replicate Figure 3 from the lab handout. A histogram of photon count rates is overlapped with a Poisson distribution (eqn. 1).

$$P(x;\mu) = \frac{\mu^x}{x!}e^{-\mu} \tag{1}$$

For a Poisson distribution with a mean value of μ, the standard deviation σ follows equation 2.

$$\sigma = \sqrt{\mu} \tag{2}$$

The code corresponding to this section can be found in the Appendix, section 6.1. I wrote a *poisson*() function to plot the probability mass function for a Poisson distribution, and I checked my plotting with *scipy.stats.poisson*(). I defined another function, *histogram*(), to help generate the histogram from the provided data points.

2 Mean and Standard Deviation

Part 3 of the lab requires calculating the mean and standard deviation of the given datasets. There was no file with my name, so I used the files "Noname-noname-*". The results are shown in Table 1.

For the "large" dataset, the mean is 1072.278. For a Poisson distribution with $\mu = 1072.278$, the standard deviation would be $\sigma = \sqrt{\mu} = 32.746$. The actual standard deviation of the measurement set was found to be $\sigma = 32.206$, which is about 1.7% higher than the expected value for the Poisson distribution. The large dataset thus has results consistent with the Poisson distribution.

For the "small" distribution, the mean is 2.743. For a Poisson distribution with $\mu = 2.743$, the standard deviation would be $\sigma = \sqrt{\mu} = 1.656$. The actual standard deviation of the measurement set was found to be $\sigma = 1.620$, which is about 2.2% higher than the expected value for the Poisson distribution. The small dataset thus has results consistent with the Poisson distribution, though they are less consistent than what was observed for the large dataset.

The code corresponding to this section can be found in the Appendix, section 6.2. The *mean*() and *std*() functions from NumPy were used to calculate the mean and standard deviation values.

Table 1: Means and standard deviations computed for each dataset.

Dataset	Mean	Standard Deviation
Large	1072.278	32.206
Small	2.743	1.620

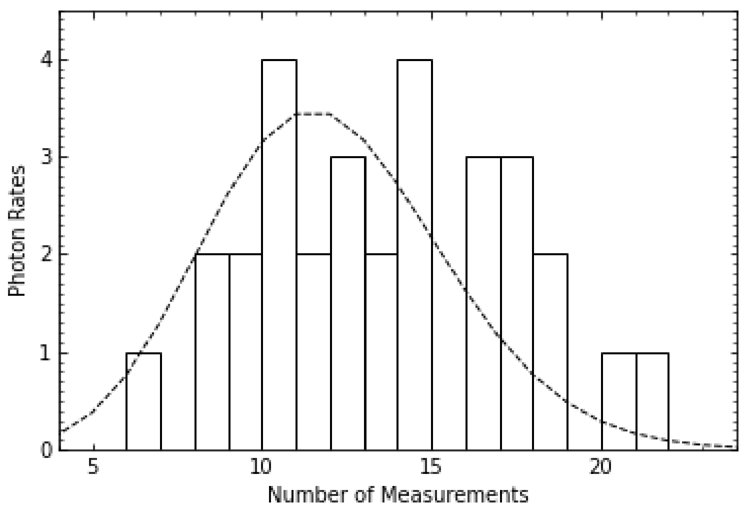

Figure 1: Histogram of photon counts measured. The solid line shows the photon count rates and the dashed line shows the expected Poisson distribution with $\mu = 12$

3 Plotting the Measurements

Part 4 of the lab requires plotting the two datasets, first in sequence and then as histograms. Each dataset has 1000 points. The code corresponding to this section can be found in the Appendix, section 6.3. The *histogram*() function from Appendix section 6.1 was called to help construct the histograms for this section.

The plotted dataset sequences are shown in Figures 2 and 4. The histograms for the datasets are shown in Figures 3 and 5.

By glancing the plots, I can roughly estimate the mean and standard deviation of the two datasets.

For the large dataset, the histogram distribution looks fairly symmetric around the highest points/maxima (Figure 3), which appear to be clustered around 1075. If I estimate the mean as 1075 and assume a Poisson distribution, then I can estimate the standard deviation as $\sqrt{1075} = 32.8$ These estimates are relatively close to values that were calculated in Table 1.

For the small dataset, it would be more difficult to estimate the mean and standard deviation of the measurement distribution. Though there is a definite peak on the histogram distribution does not look as symmetric as it did for the large dataset. From Figure 5, the he mean looks to be around 3, or slightly greater. If we assume a Poisson distribution (though Figure 5 does not look very much like a Poisson distribution) we can estimate then that the standard deviation is $\sqrt{3} = 1.73$. These values are not as relatively close to what was calculated in Table 1.

Comparing the histogram for the small dataset (Figure 5) with what is expected from a Poisson distribution as Figure 1, the distribution of the small dataset is not symmetric about its mean, which is expected of a Poisson distribution. There is a singular peak, which is expected, but the tails on either of the side of the mean decrease at different rates.

4 Poisson Distribution with Large Averages

When the expected average for a Poisson distribution is very large, the distribution becomes similar to a Gaussian distribution. The Gaussian distribution has the following probability distribution function:

$$P(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{3}$$

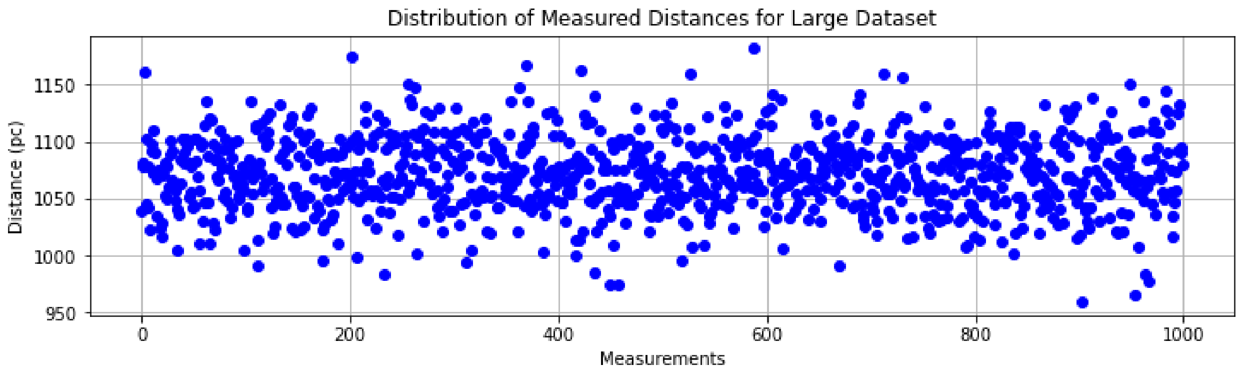

Figure 2: Plotted measurement sequence for the large dataset.

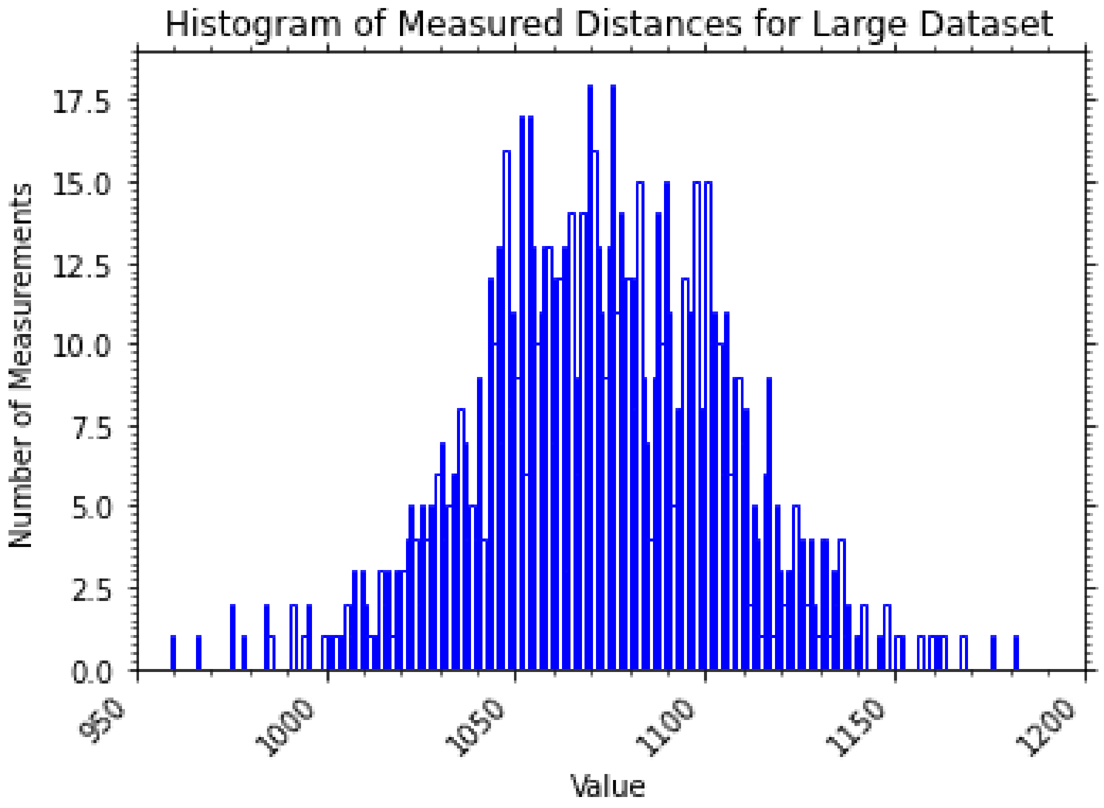

Figure 3: Histogram for the large dataset.

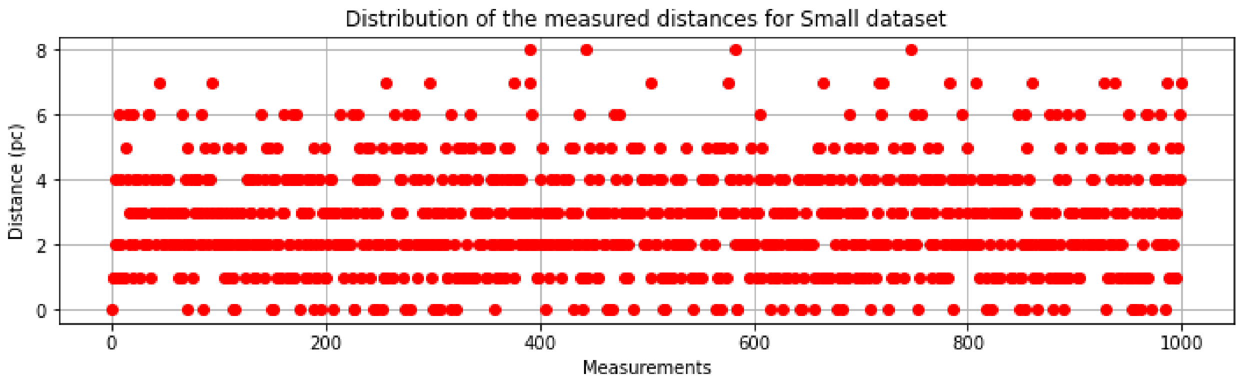

Figure 4: Plotted measurement sequence for the small dataset.

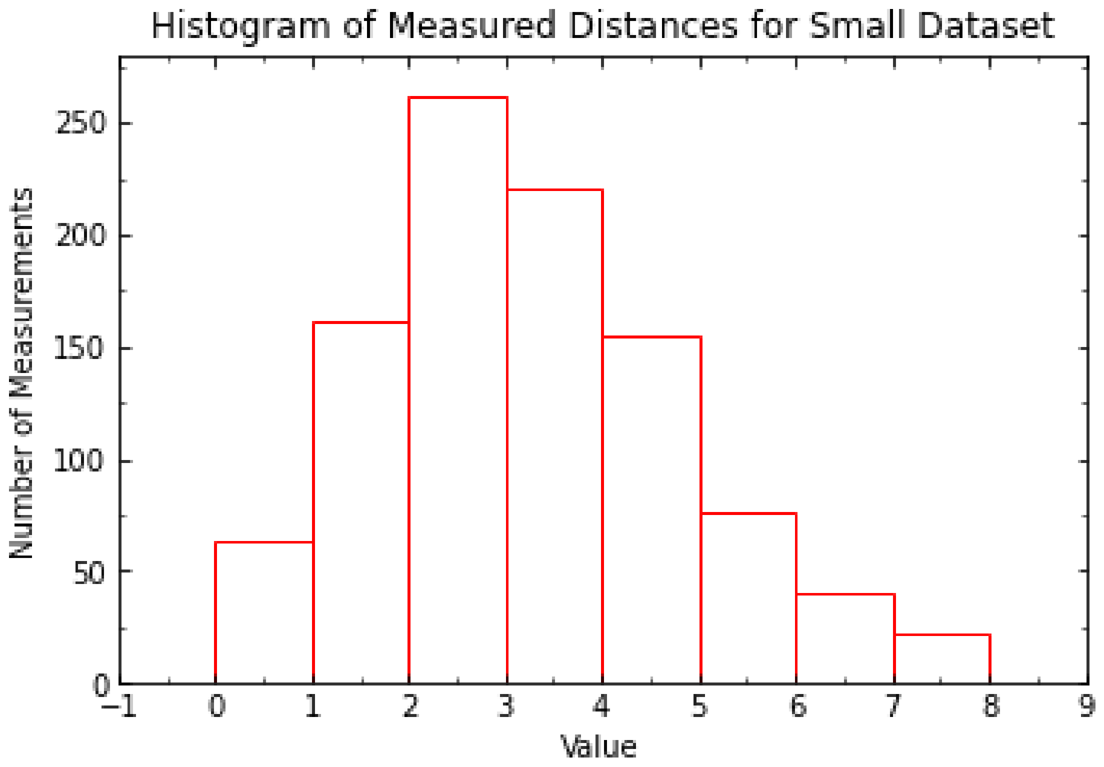

Figure 5: Histogram for the small dataset.

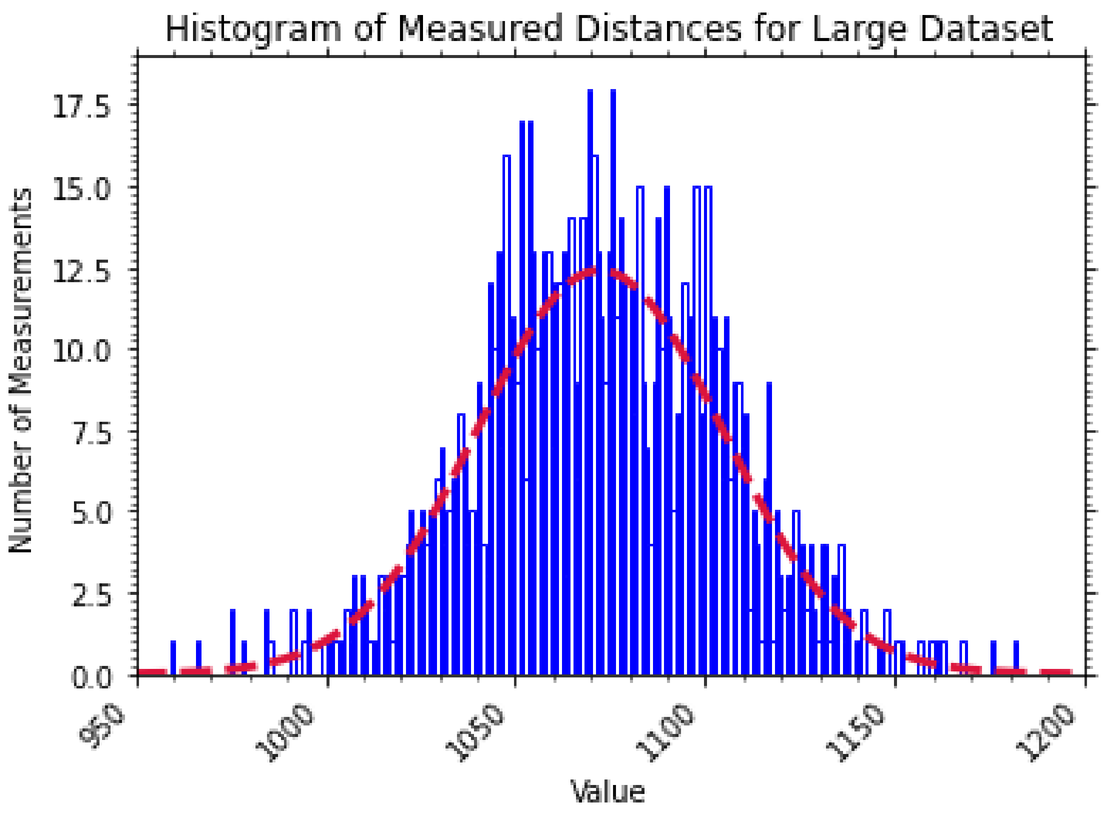

Figure 6: Histogram for the large, overplotted with the Gaussian $P(x; \mu = 1072.278, \sigma = 32.206)$.

Between the histograms for the small and large datasets, I think that the the histogram for the large measurements (Figure 3) looks more like a Gaussian. This is because the resulting histogram looks more symmetric about its mean than the histogram for the small dataset (Figure 5).

In Figures 6 and 7, the Gaussian distribution expected from the mean and standard deviation values for the large and small datasets are plotted over the histograms for each respective dataset.

The code corresponding to this section can be found in the Appendix, section 6.4. The *histogram*() function from Appendix section 6.1 was called to help construct the histograms for this section. I also defined a function *gauss*() to generate the Gaussian probability mass functions corresponding to the means and standard deviations for the provided datasets. I tested this function against *scipy.stats.norm*() to ensure that my resulting probability distributions were correct.

Figure 6 shows that the Gaussian distribution appears to underestimate the measurements for the most part, but the width of the distribution (corresponding to the standard deviation) looks well approximated. As for Figure 7, many of the values on the histogram are greater than the corresponding value on the Gaussian distribution. Since this distribution does not look as symmetric as the distribution for the large dataset, I think that, between the two datasets, the large dataset more closely resembles a Gaussian distribution.

5 Calculating the Hubble Constant

For this part of the lab, the file "Noname-noname-Hubble.txt." was used. The data file contains distances [Mpc] data for 13 galaxies within the range of [1,2000] Mpc, as well as the observed receding velocities of these galaxies [km/s].

The least squares method can be used to find the slope m and y-intercept c for a 2-dimensional set of measurements $\hat{x}$ and $\hat{y}$ (each of length N) with the following equation:

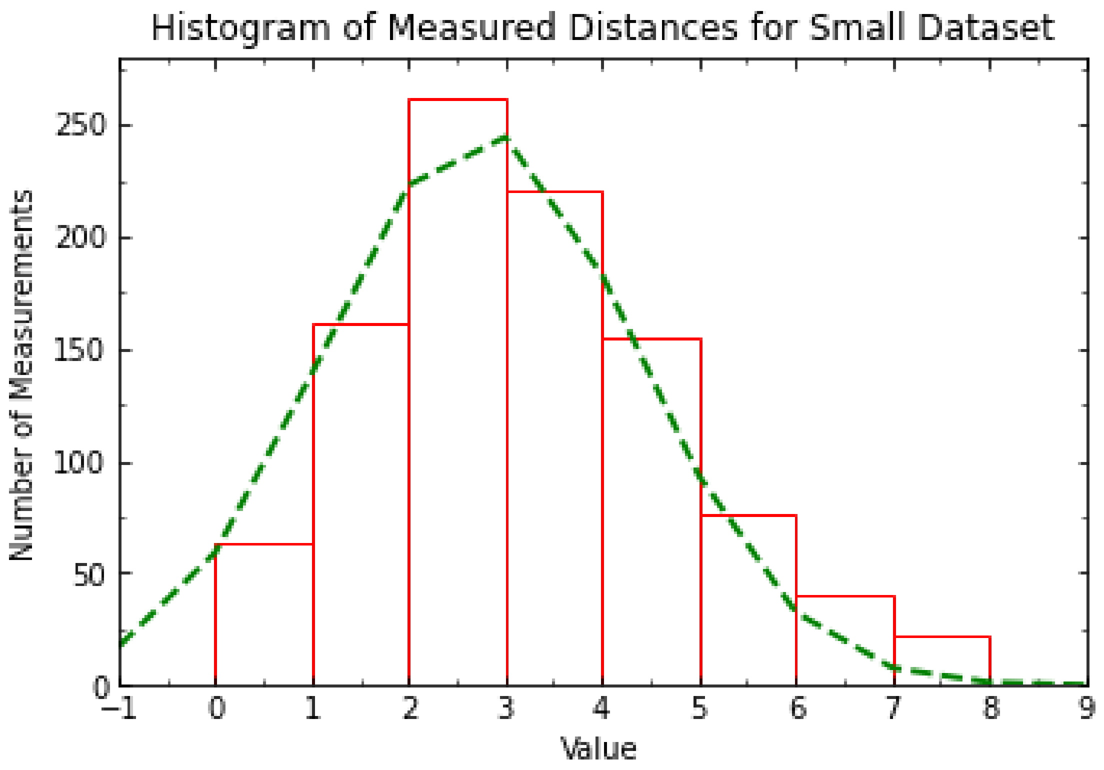

Figure 7: Histogram for the small dataset, overplotted with the Gaussian $P(x; \mu = 2.743, \sigma = 1.620)$.

$$\begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & N \end{bmatrix}^{-1} \begin{bmatrix} \sum x_i y_i \\ \sum y_i \end{bmatrix}$$

(4)

The standard deviation for the slope m, σ_m, is found in eqn. (5) and the standard deviation for the y-intercept c, σ_c, is found in eqn. (6).

$$\sigma_m^2 = \frac{N\sigma^2}{N\sum x_i^2 - (\sum x_i)^2} \tag{5}$$

$$\sigma_{c^2} = \frac{\sigma^2 \sum x_i^2}{N\sum x_i^2 - (\sum x_i)^2} \tag{6}$$

Since the standard deviation σ defined in these equations is not known, we can estimate its value by using eqn. (7).

$$\sigma^2 = \frac{1}{N-2} \sum [y_i - (mx_i + c)]^2 \tag{7}$$

I used the provided data to calculate the Hubble constant by using the linear least squares fitting method. The code corresponding to this section can be found in the Appendix, section 6.5. I defined a function $least\_squares\_fit()$ to compute the slope and y-intercept of the linear squares fit for the inputted data, along with their respective uncertainties. In this function, I estimated the standard deviation of the inputted dataset using eqn. (7).

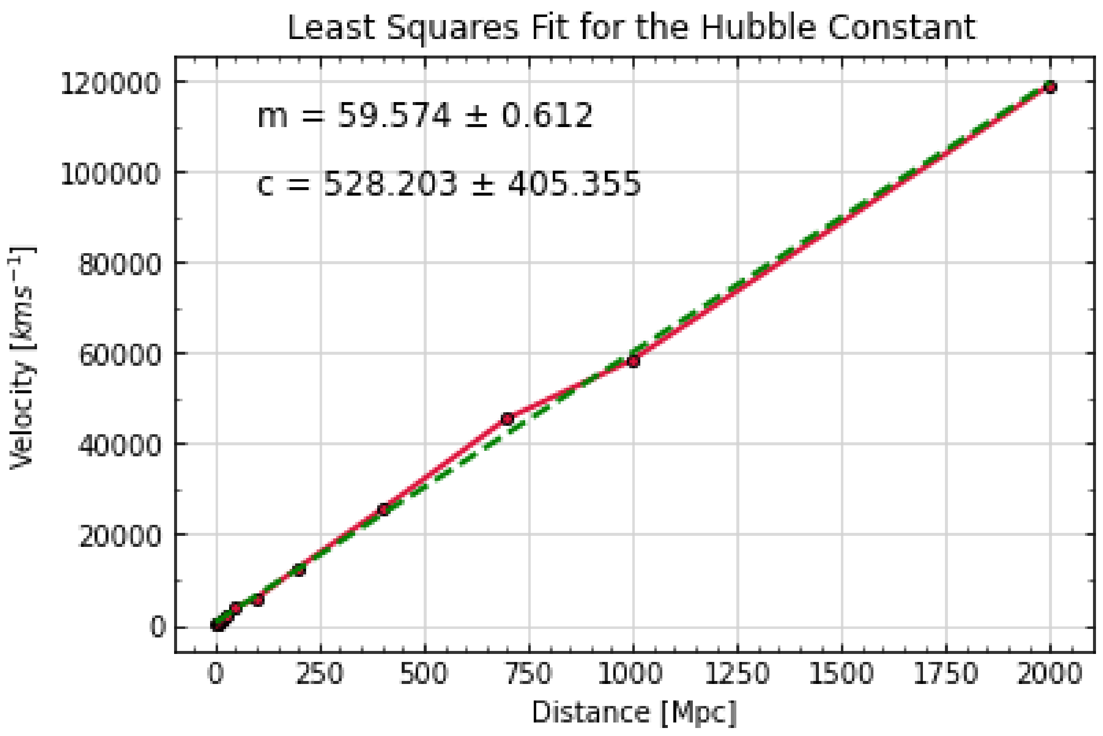

Figure 8: Determining the Hubble constant by using the least squares fitting method.

From using least squares fitting, the Hubble constant was found to be 59.574 ± 0.612 km s^{-1}/Mpc. A plot of the data and the least squares fit is shown in Figure 8. The y-intercept of the fit was found to be 528.203 ± 405.355 km s^{-1}. The uncertainty for this value is relatively much higher than the uncertainty of the slope. This makes sense though, because we are expecting, from Hubble's Law, that the data should fit a straight line with a y-intercept of zero. This is why the uncertainty of the y-intercept we found has the same order of magnitude as its uncertainty.

Table 2: Results of least squares regression for the Hubble constant.

m [km s^{-1}/Mpc]	σ_m [km s^{-1}/Mpc]	σ_m^2 [km s^{-1}/Mpc]	c [km s^{-1}]	σ_c [km s^{-1}]	σ_c^2 [km s^{-1}]
59.574	0.612	0.375	528.203	405.355	164312.822

6 APPENDIX: Python Code

The code used in this lab was merged in one .py file, but to check the code for each question, I have split the code into parts.

6.1 Reproducing The Poisson Distribution

```
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator
import numpy as np
import scipy.special
from scipy.stats import poisson, norm
```

```
# Part 1 of the Lab requires recreating the histogram figure shown in the lab handout.
def get_histogram(points):
    """
    This function returns an  array of each of the unique values in the input
    points array as well the number of occurrences each value had in the input array

    Parameters
    ----------
    points : array
        Array of ints.

    Returns
    -------
    values : array
        Array of unique values found in the input points array
    counts : array
        Array of number of occurrences of each of the values in the input points array

    """

    values = []
    counts = []
    for i in points:
        # Check if already in possible values list
        if i not in values:
            values += [i]
            counts += [1]
        else:
            index = values.index(i)
            counts[index] += 1

    return values, counts

def poisson(x_points, mean):
    """
    This function gives the Poisson distribution for an input array of x values.

    Parameters
    ----------
    x_points : array
        x-axis points for a Poisson distribution.
    mean : float
        mean value for the Poisson distribution.

    Returns
    -------
    y_points : array
        Poisson distribution for input x_points array

    """

    a = mean**x_points
    b = scipy.special.factorial(x_points)
    c = np.divide(a, b)
    d = np.exp(-mean)

    return np.multiply(c, d)

data_points = np.array([13, 17, 18, 14, 11, 8, 21, 18, 9, 12, 9, 17, 14,
                        6, 10, 16, 16, 11, 10, 12, 8, 20, 14, 10, 14, 17, 13, 16, 12, 10])
n_points = len(data_points)
bins = np.linspace(4, 25, 22)
values, counts = get_histogram(data_points)

# Calculate Poisson probablilities with an average of 12
x = np.linspace(0, 29, 30)
# Normalize the probabilities for the results we have produced
y = poisson(x, 12)
y = y/np.sum(y)
# Then, multiply by no. of data points)
y = y*30

# y = scipy.stats.poisson.pmf(x, mu=12)
# y = y/np.sum(y)
# # Then, multiply by no. of data points)
# y = z*30

plt.hist(values, bins=bins, weights=counts, facecolor="white", edgecolor="black")
plt.plot(x, y, linestyle="--", color="black", linewidth=1)
plt.xlabel("Number of Measurements")
plt.ylabel("Photon Rates")
plt.xticks(np.linspace(5,25,5))
plt.yticks(np.linspace(0,4,5))
plt.xlim(4,24)
plt.ylim(0,4.5)
ax = plt.gca()
ax.xaxis.set_minor_locator(MultipleLocator(1))
ax.yaxis.set_minor_locator(MultipleLocator(0.1))
ax.tick_params(direction="in", which="both", top=True, bottom=True, left=True, right=True)

plt.savefig("AST325_Lab1_Fig1.png")
plt.show()
```

6.2 Mean and Standard Deviation

```
# Part 3 of the lab requires calculating the mean and standard deviation of the given data
# There was no file with my name, so I used the file  Noname -noname*.
small_file = "Noname-noname-Small.txt"
large_file = "Noname-noname-Large.txt"

small_data = np.loadtxt(small_file)
large_data = np.loadtxt(large_file)

small_mean = np.mean(small_data)
small_stddev = np.std(small_data)
```

```
large_mean = np.mean(large_data)
large_stddev = np.std(large_data)

print("Small Dataset Mean:", small_mean)
print("Small Dataset Standard Deviation:", small_stddev)
print("Large Dataset Mean:", large_mean)
print("Large Dataset Standard Deviation:", large_stddev)
```

6.3 Plotting the Measurements

```
# Part 4 of the lab requires plotting the datasets, first in sequence and # then as a histo

# Make plots of measurement sequences
# Small dataset
plt.figure(figsize=(12, 3))
plt.plot(small_data, linestyle='None', color='red', marker='o')
plt.title("Distribution of the measured distances for Small dataset")
plt.xlabel("Measurements")
plt.ylabel("Distance (pc)")
plt.grid(visible=True)
plt.savefig("AST325_Lab1_Small_Sequence.png")
plt.show()

# Large Dataset
plt.figure(figsize=(12, 3))
plt.plot(large_data, linestyle='None', color='blue', marker='o')
plt.title("Distribution of Measured Distances for Large Dataset")
plt.xlabel("Measurements")
plt.ylabel("Distance (pc)")
plt.grid(visible=True)
plt.savefig("AST325_Lab1_Large_Sequence.png")
plt.show()

# Make plots for histograms of each measurement set
s_min, s_max = int(min(small_data)), int(max(small_data))
l_min, l_max = int(min(large_data)), int(max(large_data))
bins_small = np.linspace(s_min,s_max,s_max-s_min+1)
bins_large = np.linspace(l_min,l_max,l_max-l_min+1)
small_values, small_counts = get_histogram(small_data)
large_values, large_counts = get_histogram(large_data)

# Small dataset
plt.hist(small_values, bins=bins_small, weights=small_counts, facecolor="white", edgecolor=
plt.title('Histogram of Measured Distances for Small Dataset')
plt.xlabel("Value")
plt.ylabel("Number of Measurements")
plt.xlim(-1,9)
plt.ylim(0,280)
ax = plt.gca()
ax.yaxis.set_minor_locator(MultipleLocator(25))
ax.xaxis.set_minor_locator(MultipleLocator(0.5))
ax.xaxis.set_major_locator(MultipleLocator(1.0))
ax.tick_params(direction="in", which="both", top=True,
                            bottom=True, left=True, right=True)
plt.savefig("AST325_Lab1_Small_Histogram.png")
plt.show()

# Large dataset
plt.hist(large_values, bins=bins_large, weights=large_counts, facecolor="white", edgecolor=
plt.title('Histogram of Measured Distances for Large Dataset')
plt.xlabel("Value")
plt.ylabel("Number of Measurements")
plt.xlim(950,1200,250)
plt.ylim(0,19)
plt.xticks(rotation=45, ha='right')
ax = plt.gca()
ax.yaxis.set_minor_locator(MultipleLocator(0.25))
ax.xaxis.set_minor_locator(MultipleLocator(10))
ax.tick_params(direction="out", which="both", top=True,
                            bottom=True, left=True, right=True)
plt.savefig("AST325_Lab1_Large_Histogram.png")
plt.show()
```

6.4 Poisson Distribution with Large Averages

```
# Part 5 of the lab requires us to overplot the Gaussian distribution expected
# from the mean and standard deviation found in Part 3 on the histograms that
# created in Step 4.

def gauss(x_points, mean, stddev):
    """
    This function gives the Gaussian distribution for an input array of x values.

    Parameters
    ----------
    x_points : array
        x-axis points for the Gaussian distribution.
    mean : float
        Mean of the Gaussian distribution.
    stddev : float
        Standard deviation for the Gaussian distribution.

    Returns
    -------
    y_points : array
        Gaussian distribution for input x_points array

    """

    a = 1/(stddev*np.sqrt(2*np.pi))
    b = ((x_points - mean)/stddev)
    c = np.multiply(b, b)
    d = np.exp(-0.5*c)

    return a*d
```

```
n_small, n_large = 1000, 1000
x_small, x_large = np.linspace(-1,999,n_small), np.linspace(950,1950,n_large)
y_large = gauss(x_large, large_mean, large_stddev)
y_small = gauss(x_small, small_mean, small_stddev)

# With SciPy functions
y_small = scipy.stats.norm(loc=float(small_mean), scale=float(small_stddev)).pdf(x_small)
y_large = scipy.stats.norm(loc=float(large_mean), scale=float(large_stddev)).pdf(x_large)
# Normalize the probabilities for the results we have produced
y_small = y_small/np.sum(y_small)
y_large = y_large/np.sum(y_large)
# Then, multiply by no. of data points)
y_small = y_small*n_small
y_large = y_large*n_large

# Small dataset
plt.hist(small_values, bins=bins_small, weights=small_counts, facecolor="white", edgecolor=
plt.plot(x_small, y_small, linestyle="--", color="green", linewidth=2)
plt.title('Histogram of Measured Distances for Small Dataset')
plt.xlabel("Value")
plt.ylabel("Number of Measurements")
plt.xlim(-1,9)
plt.ylim(0,280)
ax = plt.gca()
ax.yaxis.set_minor_locator(MultipleLocator(25))
ax.xaxis.set_minor_locator(MultipleLocator(0.5))
ax.xaxis.set_major_locator(MultipleLocator(1.0))
ax.tick_params(direction="in", which="both", top=True,
               bottom=True, left=True, right=True)
plt.savefig("AST325_Lab1_Small_Histogram_Gauss.png")
plt.show()

# Large dataset
plt.hist(large_values, bins=bins_large, weights=large_counts, facecolor="white", edgecolor=
plt.plot(x_large, y_large, linestyle="--", color="crimson", linewidth=3)
plt.title('Histogram of Measured Distances for Large Dataset')
plt.xlabel("Value")
plt.ylabel("Number of Measurements")
plt.xlim(950,1200,250)
plt.ylim(0,19)
plt.xticks(rotation=45, ha='right')
ax = plt.gca()
ax.yaxis.set_minor_locator(MultipleLocator(0.25))
ax.xaxis.set_minor_locator(MultipleLocator(10))
ax.tick_params(direction="out", which="both", top=True,
               bottom=True, left=True, right=True)
plt.savefig("AST325_Lab1_Large_Histogram_Gauss.png")
plt.show()
```

6.5 Calculating the Hubble Constant

```
# Part 6 of the lab requires calculating the Hubble constant using the straight-line
# linear least squares fitting.
```

```
def least_squares_fit(x,y):
    """
    This function gives the slope and y intercept of a 2-array dataset for a least squares

    Parameters
    ----------
    x : 1D array of floats or ints
        x-axis points.
    y : 1D array of floats or ints
        Mean of the Gaussian distribution.

    Returns
    -------
    m : float
        Resulting slope for least squares fitting.
    c : float
        Resulting y-intercept for least squares fitting.
    sigma_m : float
        The standard deviation of the slope m.
    sigma_c : float
        The standard deviation of the slope m

    """
    # Define variables for slope and y intercept
    m, c, sigma_m, sigma_c = 0, 0, 0, 0

    length = np.size(x)
    if length == np.size(y): # Check that input arrays are the same length
        # Construct the matrices, [m c] = A^-1 * B
        # Start with matrix A
        a, b, d, e = 0, 0, 0, length
        for i in range(0, length, 1):
            a += x[i]**2
            b += x[i]
            d += x[i]
        A = np.array([[a, b], [d, e]])
        # Get the inverse of matrix A
        A_inv = np.linalg.inv(A)

        # Construct matrix B
        f, g = 0, 0
        for i in range(0, length, 1):
            f += x[i]*y[i]
            g += y[i]
        B = np.array([[f], [g]])

        # Find m and c resulting from [m c] = A_inv * B
        h, j, k, l = A_inv[0,0], A_inv[0,1], A_inv[1,0], A_inv[1,1]
        m = h*f + j*g
        c = k*f + l*g
        # Alternatively, can use numpy.matmul()
        # m, c = np.matmul(A_inv, B)
        # Now find the errors for the slope and y intercept
        # First estimate the standard deviation sigma
        C = 0
        for i in range(0, length, 1):
            C += (y[i] - (m*x[i] + c))**2
        sigma_2 = (1/(length-2))*C
        sigma = np.sqrt(sigma_2)

        # Now find sigma_m
        sigma_m_2 = length*sigma_2 / ( length*a - (sum(x))**2 )
        sigma_m = np.sqrt(sigma_m_2)

        # Now find sigma_c
        sigma_c_2 = sigma_2*a / ( length*a - (sum(x))**2 )
        sigma_c = np.sqrt(sigma_c_2)

        print("Slope m [km s^-1 / Mpc]:", m)
        print("Slope m [km s^-1 / Mpc] variance,  ^2_m:", sigma_m_2 )
        print("Slope m [km s^-1 / Mpc] standard deviation,  _m :", sigma_m)
        print("Y-intercept m [km s^-1]:", c)
        print("Y-intercept [km s^-1] variance,  ^2_c:", sigma_c_2 )
        print("Y-intercept [km s^-1] standard deviation,  _c :", sigma_c)

    return np.array([m, c, sigma_m, sigma_c])

hubble_file = "Noname-noname-Hubble.txt"
hubble_data = np.loadtxt(hubble_file)

distances = hubble_data[:,0]
velocities = hubble_data[:,1]
mc = least_squares_fit(distances, velocities)
m, c, sigma_m, sigma_c = mc[0], mc[1], mc[2], mc[3]

plt.plot(distances, velocities,
         linestyle="-", marker="o", markeredgecolor="black", color="crimson",
         linewidth=2, markersize=4, label="Data")
slope_text = "m = " + "{:.3f}".format(m)
const_text = "c = " + "{:.3f}".format(c)
plt.text(100, 1.10e5, slope_text, fontsize="large")
plt.text(100, 0.95e5, const_text, fontsize="large")
plt.plot(distances, distances*m+c, linestyle="--", color="green", linewidth=2, label="OLS
plt.title('Least Squares fit for the Hubble Constant')
plt.xlabel("Distance [Mpc]")
plt.ylabel("Velocity [$km s^{-1}$]")
ax = plt.gca()
ax.yaxis.set_minor_locator(MultipleLocator(1e4))
ax.xaxis.set_minor_locator(MultipleLocator(50))
ax.tick_params(direction="in", which="both", top=True,
                              bottom=True, left=True, right=True)
plt.grid(color="lightgrey")
plt.savefig("AST325_Lab1_Hubble.png")
plt.show()
```