EAX - Error Analysis Exercise

Physics 111B Fall 2024

Note

All Physics 111 Advanced Lab students are required to complete this assignment at the beginning of the semester. It will be graded on 50 points basis; a late turn-in is allowed only with the instructor's approval before the due date. Don't jeopardize your grade on your first experiment by being late with this assignment. You need to know how to calculate uncertainty in your data before you start a laboratory experiment.

Important: View the video introduction to error analysis (you need to use your Berkeley email to access this). The Error Analysis Exercise due date is listed under Advanced Lab Report Due Dates.

References

[Books available online with UC Berkeley authentication]

1. P. Bevington, **"Data Reduction and Error Analysis for the Physical Sciences",** McGraw-Hill. [An old standard that is pretty dry but straightforward. Chapter 5 is particularly important.]
2. A. C. Melissinos and J. Napolitano, [**"Experiments in Modern Physics, 2nd Edition"**], Academic Press (2003).
3. W. H. Press, et al., [**"Numerical Recipes in C**:] The Art of Scientific Computing, 2nd Edition", Cambridge University Press (1992); refer to Ch. 14—"Modeling of Data". [The Numerical Recipes in Pascal or FORTRAN books contain identical information. This book is the standard reference for doing scientific work on computers. Chapter 14 has a good introduction to the method of maximum likelihood, chi–square fitting, modeling data in general, error estimates of fit parameters, and, important for later experiments, the Monte Carlo method of simulation.]
4. I. G. Hughes and T. P. A. Hase, **Measurements and their Uncertainties**, Oxford University Press (2010). [This is a well-written thin book that covers many basic concepts of statistics, extremely useful for this course.]
5. Louis Lyons, "**A Practical Guide to Data Analysis for Physical Science Students"** (1991) Cambridge Press; QC33.L9 1991

6. Yardley Beers, **"Introduction to the Theory of Error"**; ADDISON-WESLEY PUBLISHING (1957) QA275 B4 1957;

Reprints and other information can be found on the Physics 111 Library Site.

Introduction

In the 111-lab, the experiment does not end when you have finished collecting your data. In many labs, you will be required to perform a detailed analysis of the data you have acquired. The point of any scientific experiment is to make quantitative statements about the properties of the physical world. A common question is, are your measurements consistent with a particular theory or not? This question can only be answered by careful analysis, including both systematic uncertainties and statistical error.

The goals of this exercise are twofold. One is to familiarize students with the basics of error analysis. Ideally, this will serve as a guide during the acquisition and analysis of data throughout the advanced lab. The second goal is to introduce students to the Python computing environment, which you will be using throughout the semester.

Before starting on EAX, please look over either the Python Tutorials https://github.com/avirukt/intro_python or the Intro to Matlab section. Additional resources for using Python are http://pythontutor.com/ and https://datahub.berkeley.edu/

Problem Set

Problem 1: Poisson statistics

We want to measure the activity (number of decays per second, a unit known as Becquerel) of a radioactive source so that we can use it to calibrate the equipment of the gamma-ray experiment. We use an electronic counter and a timer to measure the number of decays in a given time interval. In round numbers we obtain 100,000 decays in 5 minutes. How long does it take (in seconds) in order to determine the activity with a statistical uncertainty of 0.2%? Explain.

Enter the calculation in the cell below. Add a "Markdown" cell to explain your calculation

For measuring decay, the lab video says that the fractional undertainty follows $\sigma_N = \frac{1}{\sqrt{N}}$, so if we have:

$$\frac{1}{\sqrt{N}} = 0.2\% \implies N = 2.5 \times 10^7$$

At 100,000 decays in 5 minutes, this means that we need to take data for 12.5 minutes to reach our desired statistiacl uncertainty.

Problem 2: Error propagation

a) You are given the measurements of two variables A and B with the associated errors σ_A and σ_B, respectively. Assuming A and B are uncorrelated, calculate the error in C, σ_C:

> (i) $C = 3A + 2B$

```
<br> (ii) $C=A/B$
<br> (iii) $C=A^B$
<br> (iv) $C=B\ln A$
```

b) In the muon lifetime experiment we obtain a histogram for the decay rate as a function of the time after the muon first enters the detector. We expect the distribution (the histogram) to be described by an exponential function. Rather than fitting with an exponential function, it is more convenient to plot the logarithm of the decay rate as a function of time and then fit a straight line to it. Since each data point (x_i, y_i) has a statistical error, σy_i, associated with it, what qualitatively happens to the shape of the errorbars when the semi-log histogram $(x_i, \log y_i)$ is plotted? Explain and illustrate. Write an equation to describe what happens to the error bars quantitatively under the assumption that y_i is reasonably large.

c) In a separate experiment, you find that $\log_{10}\mathrm{E}_0 = 1.7 \pm 0.7$ (at 68% confidence level, CL). What is the value of E_0 and the experimental bounds at 68% CL? (Note that 0.7 is not small compared to 1.7).

Write down the calculation in the "Markdown" cell below, or enclose a separate file. Show the details of your calculation.

For part (a), we know that the errors add as follows:

$$\sigma_C = \sqrt{\left(\frac{\partial C}{\partial A}\right)^2 \sigma_A^2 + \left(\frac{\partial C}{\partial B}\right)^2 \sigma_B^2}$$

and we just add more variables/uncertainties inside the square root if we have them. Thus, we have the following:

> (i) $C = 3A + 2B$
>
> Following the formula:
>
> $$\sigma_C = \sqrt{3\sigma_A^2 + 2\sigma_B^2}$$
>
> (ii) $C = A/B$
>
> $$\sigma_C = \sqrt{\frac{\sigma_A^2}{B^2} + \frac{A\sigma_B^2}{B^4}}$$
>
> (iii) $C = A^B$
>
> $$\sigma_C = \sqrt{A^{B-1}B\sigma_A^2 + A^{2B}\sigma_B^2}$$
>
> (iv) $C = B\ln A$
>
> $$\sigma_C = \sqrt{\left(\frac{B}{A}\right)^2 \sigma_A^2 + (\ln A)^2\sigma_B^2}$$

Problem 3: Central Limit Theorem

Here we will verify the Central Limit Theorem and reproduce a plot similar to that from Wikipedia (https://en.wikipedia.org/wiki/Central_limit_theorem#/media/File:Dice_sum_central_limit_theorem.svg)

a) Write a function that returns n integer random numbers, uniformly distributed between 1 and 8, inclusively. This represents n throws of a fair 8-sided die. The value that comes up at each throw will be called the "score".

b) Generate a distribution of 1000 dice throws and plot it as a histogram normalized to unit area. Compute the mean μ_1 and standard deviation σ_1 of this distribution. Compare your numerical result to the analytical calculation.

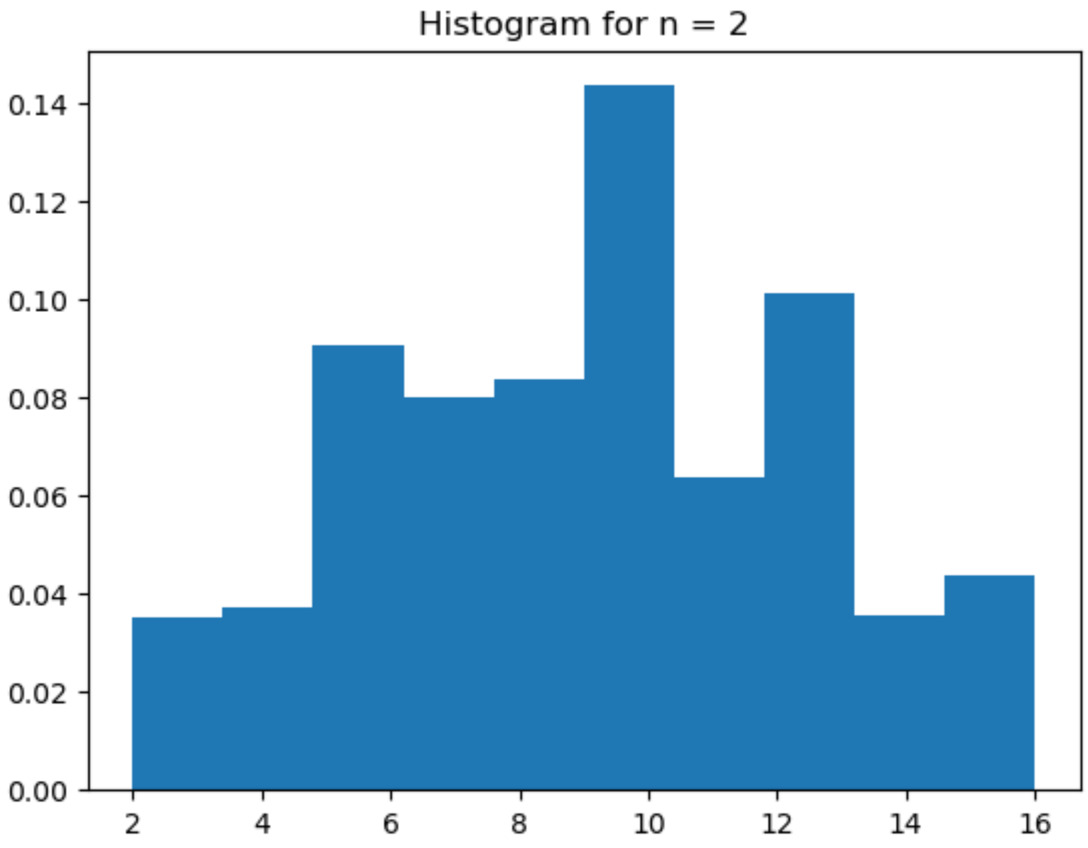
c) Generate 1000 sets of throws of $N = 2, 3, 4, 8, 16, 32, 64$ dice, computing the total sum of dice scores for each set. For each value of N, plot the distribution of total scores, and compute the mean μ_N and standard deviation σ_N of each distribution. This should be similar to the plot at the link above.

d) Plot the standard deviation σ_N as a function of N. Does it follow the Central Limit Theorem?

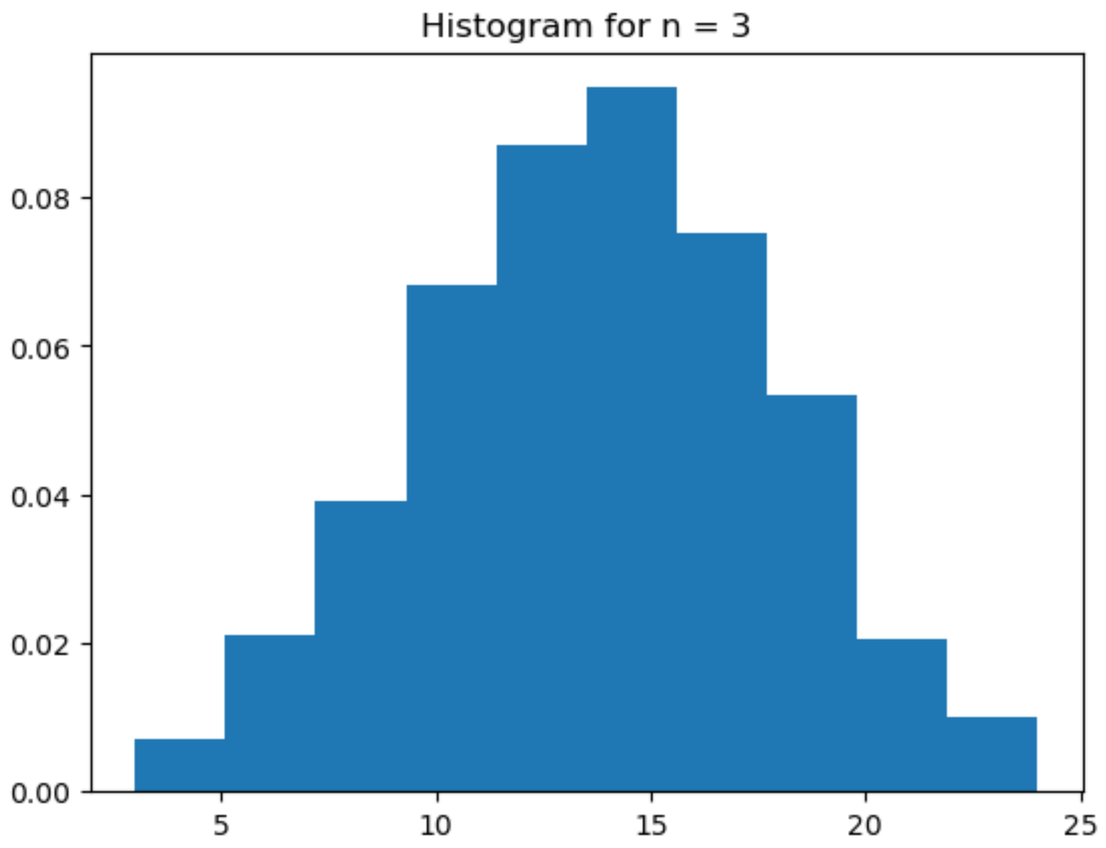
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# part (a)
score = np.random.randint(1, 9)

# part (b)
vals = np.random.randint(1, 9, size = 1000, dtype = int)
print(f"Mean: {np.mean(vals)}, standard deviation: {np.std(vals)}")

N = [2, 3, 4, 8, 16, 32, 64]
mu = []
sigma = []
for n in N:
    sums = []
    for _ in range(1000):
        sum = np.sum(np.random.randint(1, 9, size = n))
        sums.append(sum)
    plt.figure()
    plt.hist(sums, density = True)
    plt.title(f"Histogram for n = {n}")
    plt.show()
    print(f"Mean {np.mean(sum)}, std: {np.std(sum)}")
    print("------------------------------")

    mu.append(np.mean(sums))
    sigma.append(np.std(sums))
```

```
plt.figure()
plt.plot(N, sigma)
plt.plot(np.arange(0, 64), np.sqrt(np.arange(0, 64)))
plt.show()
# plt.plot(std_values)
# plt.show()
```

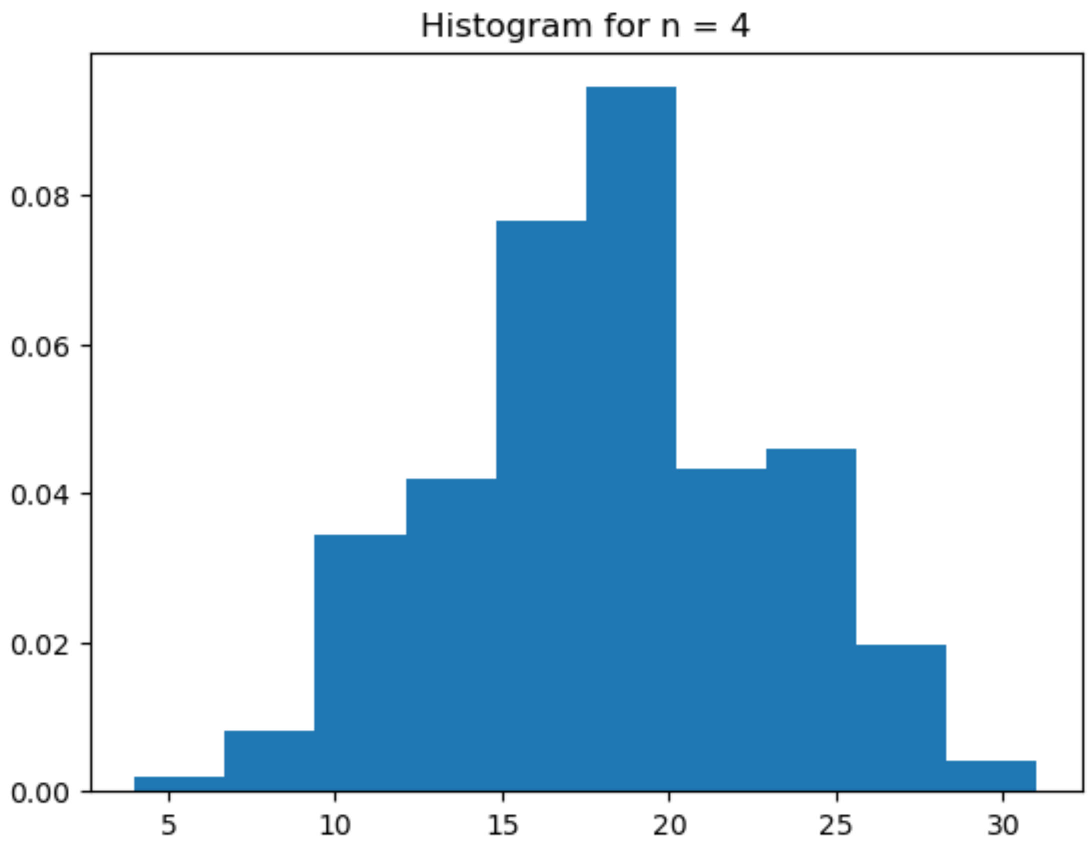
```
Mean: 4.541, standard deviation: 2.247736416931487
```



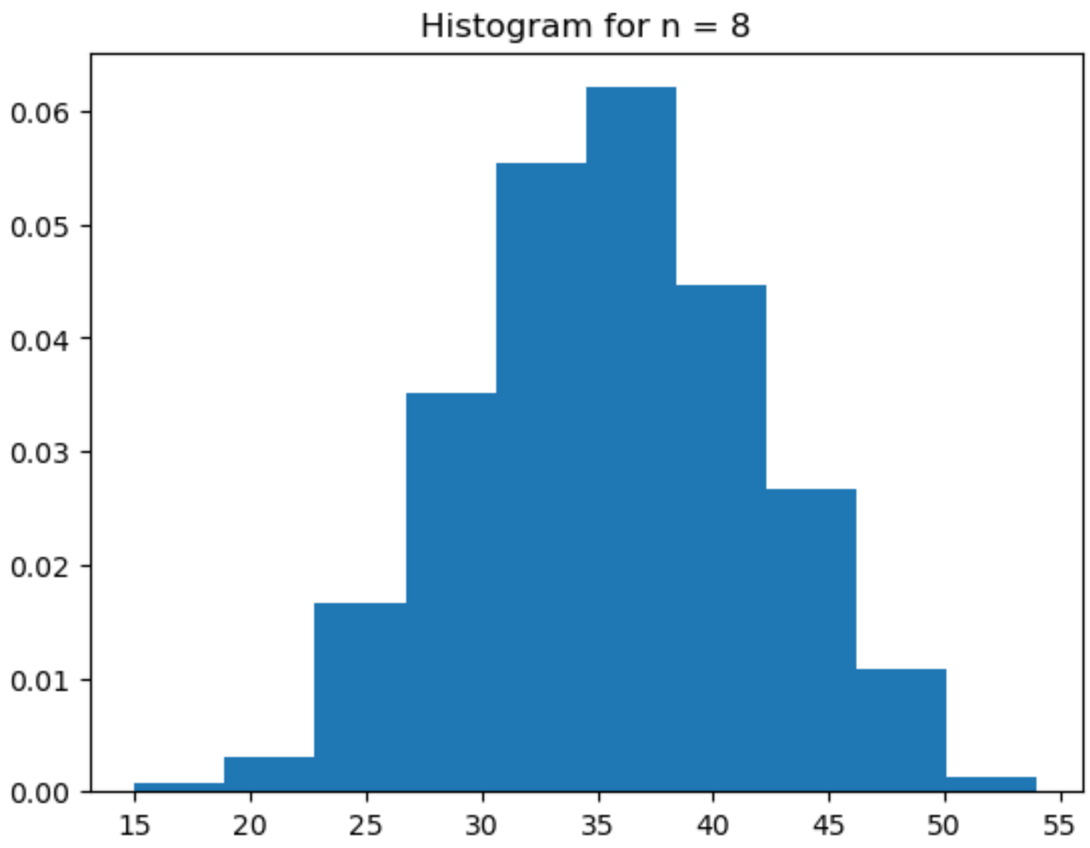
```
Mean 13.0, std: 0.0
------------------------------
```



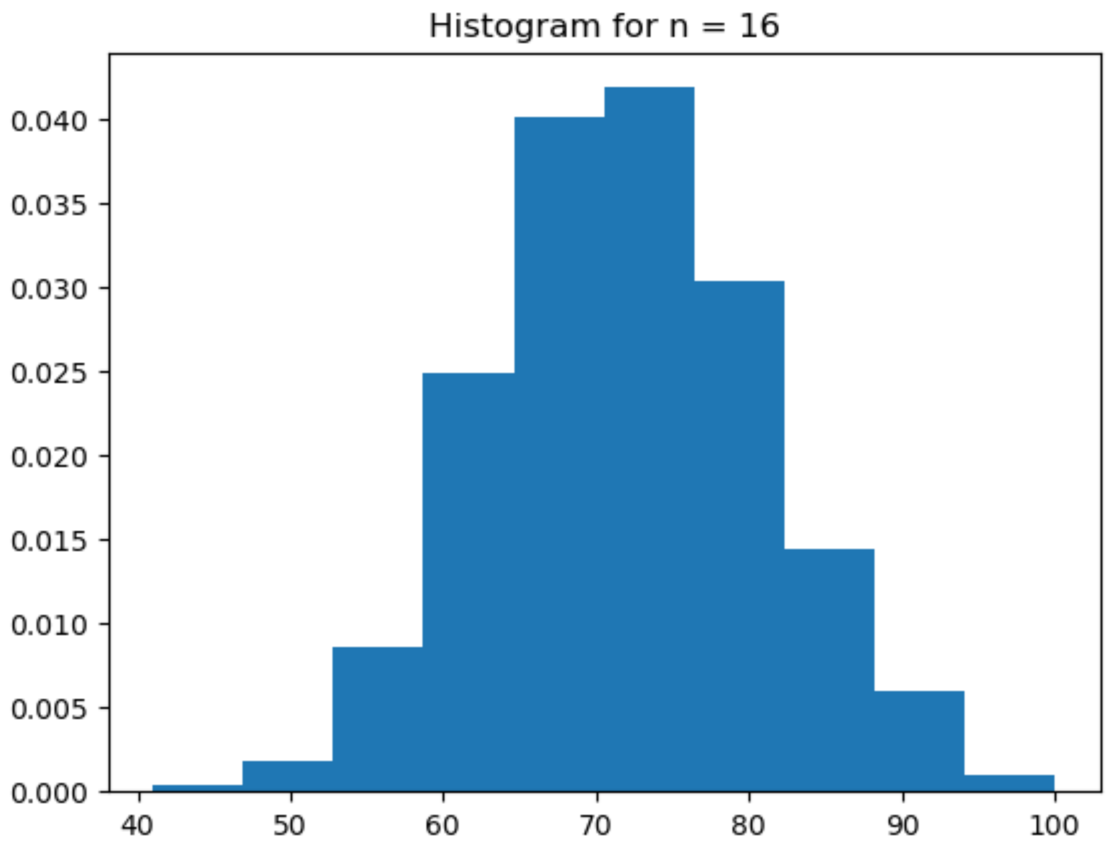
```
Mean 8.0, std: 0.0
------------------------------
```



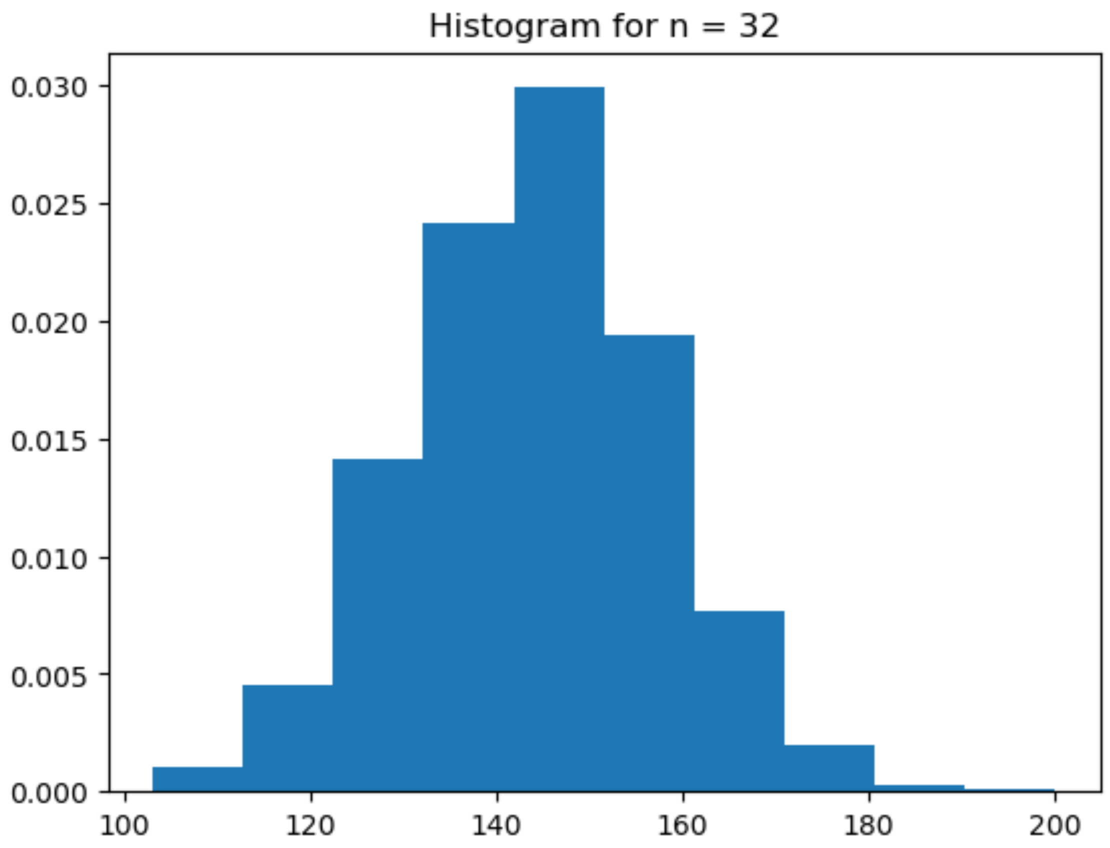
```
Mean 26.0, std: 0.0
------------------------------
```



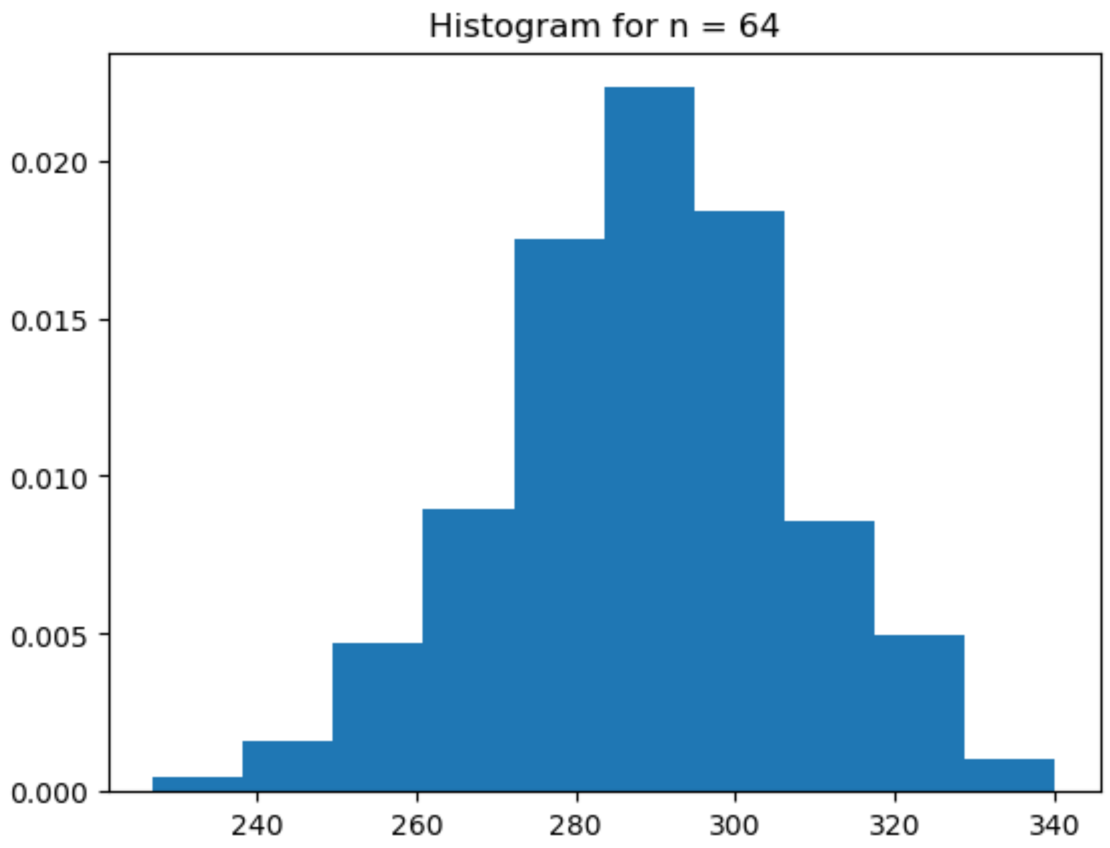
```
Mean 36.0, std: 0.0
------------------------------
```



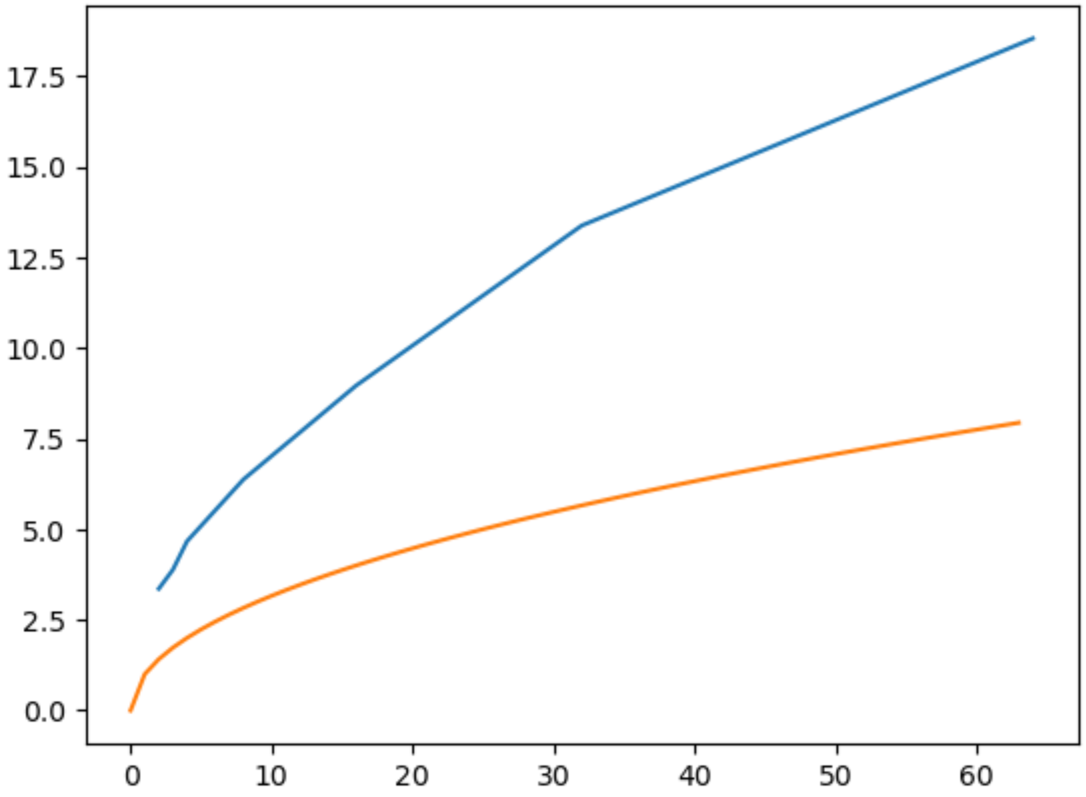
```
Mean 84.0, std: 0.0
------------------------------
```



```
Mean 148.0, std: 0.0
----------------------------
```



```
Mean 287.0, std: 0.0
------------------------------
```



From CLT, we know that $\frac{\sigma_N}{\sqrt{N}} \sim \frac{1}{\sqrt{N}} \implies \sigma_N \sim \sqrt{N}$, which is exactly the trend we see. The orange line marks $\sigma_N = \sqrt{N}$ which doesn't match our plot, but we can still clearly see that σ_N does trend like some scaled square-root function, as predicted.

Problem 4: gamma peak

You are given a dataset (peak_f24.dat), in the current directory) from a gamma-ray experiment consisting of ~1000 detections. Each line in the file corresponds to one recorded gamma-ray event and stores the measured energy of the gamma-ray. We will assume that the energies are randomly distributed about a common mean, and that each event is uncorrelated to others. Read the dataset from the enclosed file and:

a) Compute the mean and standard deviation of the given distribution of measured energies. These are estimates of the mean and standard deviation of the underlying parent distribution. Determine the uncertainties for these estimates of the mean and variance of the parent distribution. (Hint: The fractional uncertainty in the estimate of the standard deviation for Gaussian distributed data is given by Equation 2.8 in Reference 4, I. G. Hughes and T. P. A. Hase, "Measurements and their Uncertainties". For the purposes of this calculation, you can assume the sample data is drawn from a Gaussian distribution).

b) Fit the distribution to a Gaussian function using an unbinned fit (*Hint:* use `scipi.stats.norm.fit()` function), and compare the parameters of the fitted Gaussian with the mean and standard deviation computed in (a).

c) Produce a histogram of the distribution of energies. Choose the number of bins wisely, i.e. so that the width of each bin is smaller than the width of the peak, and at the same time so that the number of entries in the most populated bin is relatively large. Plot error bars on this histogram representing the uncertainty on the number of counts in each bin. These will serve as the weights for the bins in fits to this distribution.

d) Overplot the binned histogram (including bin error bars) with the best fit Gaussian function from part (b).

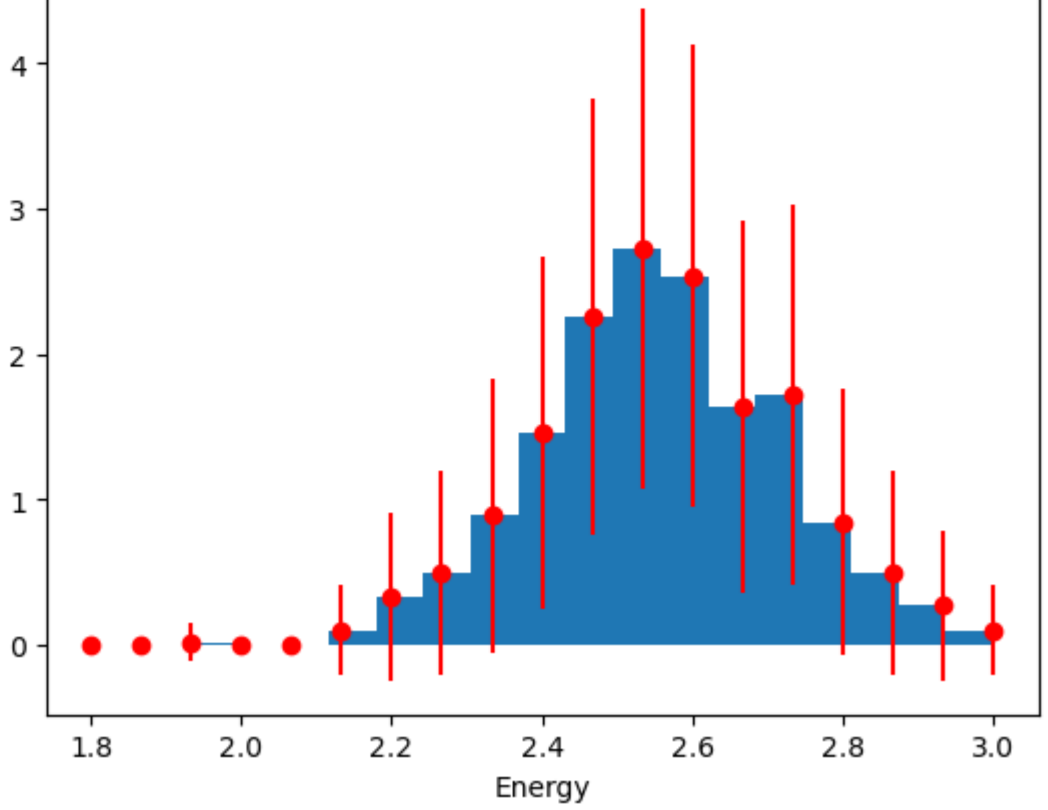
e) Fit the binned distribution to a Gaussian function using a binned least-squares fit (*Hint:* use `scipy.optimize.curve_fit()` function) and compare the parameters of the fitted Gaussian and their uncertainties to the parameters obtained in the unbinned fit in part(a).

f) How consistent is the binned distribution with a Gaussian? In other words, compare the binned histogram from part (c) to the best fit Gaussian parameters from both the unbinned and binned fits in parts (b) and (e) and compute a goodness-of-fit value, such as χ^2/dof for each of the fits.

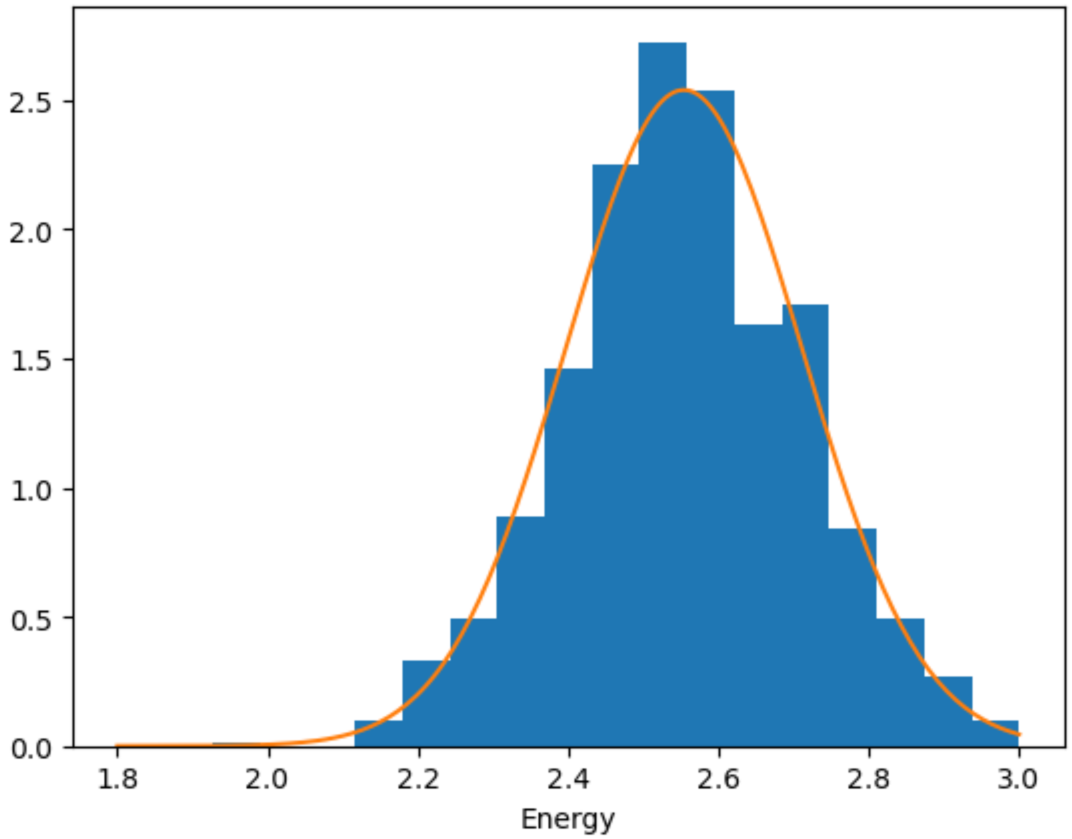
In [ ]:
```
import scipy
import scipy.optimize as opt

data = np.loadtxt('peak_f24.dat', dtype = float)

print("Part (a)")
print(f"Mean: {np.mean(data)}, Standard deviation: {np.std(data)}")
print(f"Uncertainty in the mean:
      {np.std(data[:-1])/np.sqrt(len(data))},
      Uncertainty in the std: {1/np.sqrt(2 * len(data) - 2)}")
print("\n")

# Part (b)
fit = scipy.stats.norm.fit(data)
print("Part (b)")
print("=============== UNBINNED FIT ================")
print(f"Fitted mean: {fit[0]}, Fitted std: {fit[1]}")
print("\n")

# Part (c)
print("Part (c)")
plt.hist(data, bins=np.linspace(1.8, 3, 20), density=True)
hist_count = np.histogram(data, bins = np.linspace(1.8, 3, 20), density = True)[0]
plt.errorbar(np.linspace(1.8, 3, 19), hist_count, yerr = np.sqrt(hist_count), fmt = "ro", ls = "none")
plt.xlabel("Energy")
plt.show()

print("\nPart (d)")
# Part (d)
x = np.linspace(1.8, 3, 200)
plt.hist(data, bins=np.linspace(1.8, 3, 20), density=True)
plt.plot(x, scipy.stats.norm.pdf(x, fit[0], fit[1]))
plt.xlabel("Energy")
plt.show()

print("\n" + "Part (e)")
hist_count = np.histogram(data, bins=np.linspace(1.8, 3, 20), density=True)[0]

def gaussian(x, mu, sigma):
    return 1/np.sqrt(2 * np.pi * sigma**2) * np.exp(-(x - mu)**2 / (2 * sigma**2))

var, cov = opt.curve_fit(gaussian, np.linspace(1.8, 3, 19), hist_count)
mu_fit, sigma_fit = var
d_mu, d_sigma = np.sqrt(np.diag(cov))

print("============= GAUSSIAN LSQ FIT ============")
print(f"Fitted mean: {mu_fit} \u00b1 {d_mu}")
print(f"Fitted std: {sigma_fit}, \u00b1 {d_sigma}")


# Part (f)
unbinned, bin_edges1= np.histogram(data, density=True)
binned, bin_edges2 = np.histogram(data, bins=np.linspace(1.8, 3, 20), density=True)

# plt.plot(bin_edges1[:-1], unbinned)
# plt.plot(bin_edges2[:-1], binned)
plt.plot(np.linspace(1.8, 3, 200), scipy.stats.norm.pdf(np.linspace(1.8, 3, 200), mu_fit, sigma_fit))
```

```python
plt.plot(np.linspace(1.8, 3, 200), scipy.stats.norm.pdf(np.linspace(1.8, 3, 200), fit[0], fit[1]))
plt.hist(data, bins=np.linspace(1.8, 3, 20), density=True)
plt.xlabel("Energy")
plt.legend(['LSQ gaussian fit (binned)', 'unbinned', 'binned gaussian'])
plt.show()

hist_values = np.histogram(data, bins= np.linspace(1.8, 3, 20), density=True)[0]
chisq_bin = np.sum((hist_values -
                    gaussian(np.linspace(1.8, 3, 19),
                             mu_fit, sigma_fit))/gaussian(np.linspace(1.8, 3, 19), mu_fit, sigma_fit))
chisq_unbin = np.sum((hist_values -
                      gaussian(np.linspace(1.8, 3, 19),
                               fit[0], fit[1]))/gaussian(np.linspace(1.8, 3, 19), fit[0], fit[1]))

print("\nPart (f)")
print(f"Binned chi^2: {chisq_bin/2}, Unbinned chi^2: {chisq_unbin/2}")
```

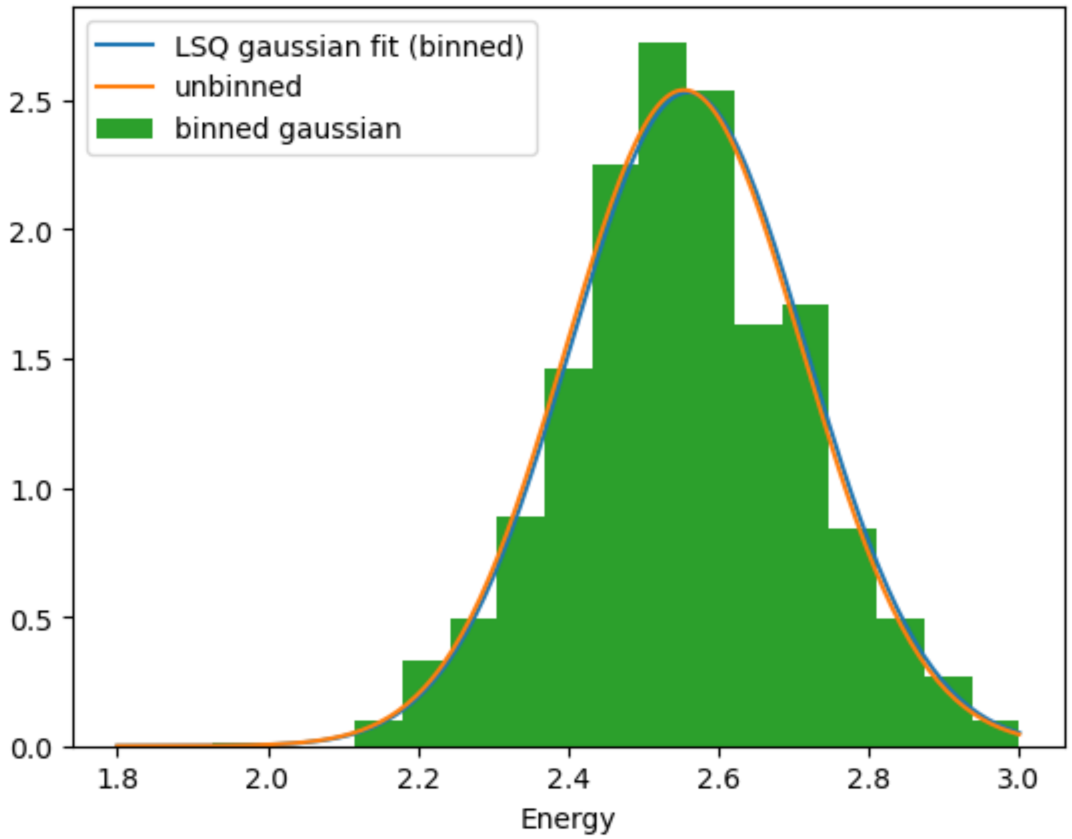
```
Part (a)
Mean: 2.5538624197499997, Standard deviation: 0.15704278151550075
Uncertainty in the mean: 0.004967725095486764, Uncertainty in the std: 0.022371868507134143


Part (b)
=============== UNBINNED FIT ================
Fitted mean: 2.5538624197499997, Fitted std: 0.15704278151550075


Part (c)
```



```
Part (d)
```



```
Part (e)
============== GAUSSIAN LSQ FIT =============
Fitted mean: 2.5595277570816126 ± 0.006492377715887679
Fitted std: 0.1576107803443869, ± 0.005303146464869677
```



```
Part (f)
Binned chi^2: 7.6312178129619825, Unbinned chi^2: 7.065971787905627
```

I'll write down my explanations here. Before going to any of the subparts, I will note that `np.histogram()` behaves a bit strangely when you try to pass in custom bins and have it compute a probability density function, which is why the density on the y-axis looks weird. I will ignore this throughout the problem, since it's not really relevant -- we only really care about the fit here.

(a) From the text, we know that the uncertainty in the mean, also called the **standard error** is given by $\alpha = \frac{\sigma_{N-1}}{\sqrt{N}}$. The uncertainty in the standard deviation is given by $\frac{1}{\sqrt{2N-2}}$.

(b) The fitted mean and standard deviations are *exactly* the same as the output from NumPy's mean and standard deviation. This would be surprising, until we realize that scipy's backend literally uses NumPy to do its computations so it's no surprise that they're also using `np.std` to get the standard deviation.

(c) The width of the peak is between 2.4 and 2.65, so we just need the bin widths to 0.2 or less. I went a bit overkill, and used bins of `np.linspace(1.8, 3, 20)`, which have a width of 0.06.

(d) The overplot for the binnned Gaussian is plotted in orange.

(e) The least squares fit returns a very similar mean and standard deviation compared to the unbinned fit, which used both `np.mean()`, `np.std()` and also `scipiy.stats.norm.fit()`. Again, from my response to part (b) we know that the similarity between the `scipy.stats.norm.fit()` and the `np.mean()` results is likely a result of both libraries using the same backend, but the fact that the least squares fit agrees is good, since this is completely independent of the two library methods.

I should also point out that we obviously *expect* them to match, since otherwise that would indicate some systematic error in our data processing.

(f) We need to divide by 2 becuase we have two degrees of freedom, μ and σ, giving us relatively good χ^2 values of 7.6 and 7.06 for binned and unbinned respectively. Obviously, we'd like for this number to be slightly lower (probably lower than 5 is best), but all things considered this isn't that bad. The χ^2 is calculated using the formula:

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i^2}$$

where O_i is the observed data points, and E_i is the expected data, so in this case, the fitted Gaussians.

Problem 5: Optical Pumping

In the optical pumping experiment (OPT), we measure the resonant frequency of a Zeeman transition as a function of the applied current (proportional to the applied magnetic field). This exercise is identical to the analysis that you will need to perform for OPT. Consider a mock data set:

Current *I* (Amps)	0.0	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0	2.2
Frequency *f* (MHz) #	0.16	0.72	1.50	2.72	3.01	3.77	5.06	5.87	6.67	6.88	7.67	8.97

a) Plot a graph of the pairs of values. Assuming a linear relationship between I and f, determine the slope and the intercept of the best-fit line using the least-squares method with equal weights, and draw the best-fit line through the data points on the graph.

b) From what they know about the equipment used to measure the resonant frequency, your lab partner hastily estimates the uncertainty in the measurement of f to be $\sigma_f = 0.18$ MHz. Estimate the probability that the straight line you found is an adequate description of the observed data if it is distributed about this line with the uncertainty guessed by your lab partner. (Hint: use scipy.stats.chi2 class to compute the quantile of the χ^2 distribution). What can you conclude from these results?

c) As in the case of part (b), we do not always have accurate a-priori estimates of the uncertainty in the data. When the data is consistent with statistical scatter about a known model, an estimate of the uncertainty in the measurements can be determined from the scatter of the data about the best fit model. Assume that the data is perfectly described by uniform statistical scatter about a linear model. Estimate the uncertainty σ_f in the measurement of f from the scatter of the data about the best fit line. Assume that all the data points have equal weight. Use this new value for σ_f to estimate the uncertainty in both the slope and the intercept of the best-fit line. This is the technique you will use in the Optical Pumping lab to determine the uncertainties in the fit parameters. This is only valid in the limit where the model is a good fit to the data and the scatter about the model is purely statistical with no additional systematic errors.

d) Now assume that the uncertainty in each value of f grows with increasing f as: $\sigma_f = 0.20\,\mathrm{MHz} + 0.02 * f$. Determine the slope and the intercept of the best-fit line using the least-squares method with these weights (weighted least-squares fit). What are the values and uncertainties for the slope and intercept? Do they differ significantly from those found for the unweighted fit? What is the resulting χ^2 for the weighted linear fit?

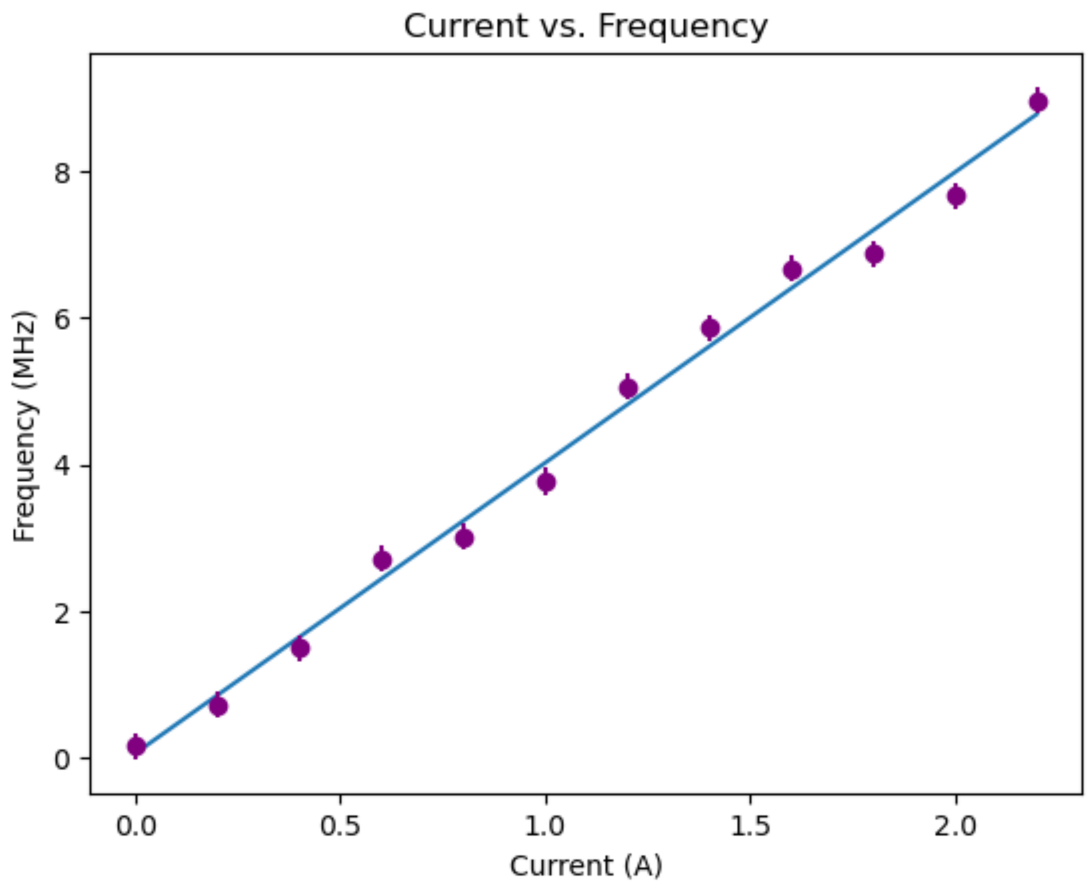
e) Here we again use the data to inform the estimate of uncertainty. Assume that a line is a good fit to the data and that the actual measurement errors are a scaled version of the function given in part (d) such that the reduced $\chi^2 = 1$ for the fit of the data to the linear model. With these assumptions, revise the estimate of uncertainties in the slope and offset of the linear fit. Plot the date with the best fit line and scaled uncertainties.

In [ ]:
```
current = np.linspace(0, 2.2, 12)
frequency = np.array([0.16, 0.72, 1.50, 2.72, 3.01, 3.77, 5.06, 5.87, 6.67, 6.88, 7.67, 8.97])

n = len(current)

# Part (a)
print("Part (a)")
```

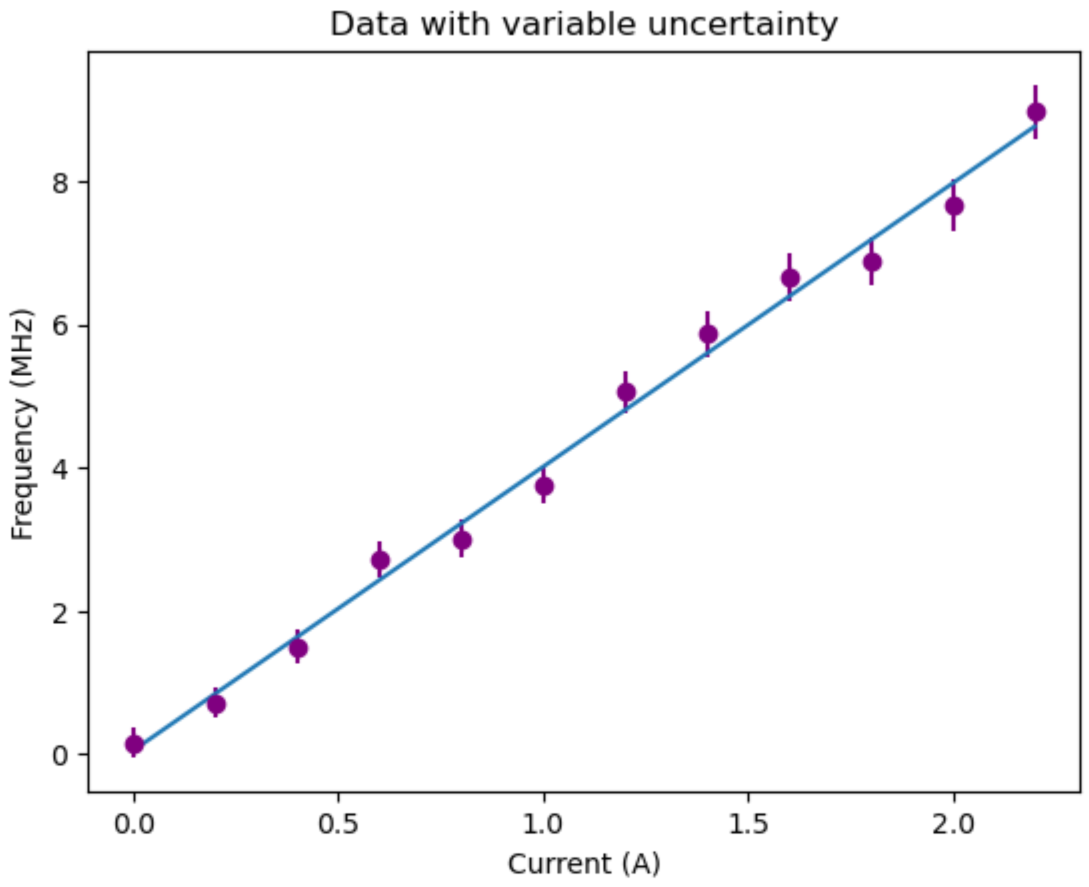
```python
m = (n*np.sum(current * frequency) -
     np.sum(current)*np.sum(frequency))/(n*np.sum(current**2)-
                                          np.sum(current)**2)
b = (np.sum(frequency) - m*np.sum(current))/n

fit_vals = m * current + b

print(f"Fitted m: {m}, b: {b}")

# Part (b)
print("\nPart (b)")
sigma_f = 0.18
ddof = len(frequency) - 2
chisq = np.sum(( (frequency - fit_vals)/(sigma_f))**2)

reduced_chi2 = chisq/ddof
p_value = scipy.stats.chi2.cdf(chisq, ddof)

print(f"P-value: {p_value}, chi^2: {chisq}, reduced chi^2: {chisq/ddof}")

# Part (c)
print("\nPart (c)")
sigma_f = np.sqrt(1 /(n - 2) * np.sum((frequency - fit_vals)**2))
sigma_m = sigma_f * np.sqrt(n/(n * np.sum(current**2) - (np.sum(current))**2))
sigma_b = sigma_f * np.sqrt(np.sum(current**2)/(n * np.sum(current**2) - (np.sum(current))**2))
print(f"m = {m} \u00b1 {sigma_m}, b = {b} \u00b1 {sigma_b}")

plt.title("Current vs. Frequency")
plt.plot(current, fit_vals)
# plt.scatter(current, frequency, c = "purple")
plt.errorbar(current, frequency, yerr = 0.18, fmt = "o", ls = "none", c = "purple")
plt.xlabel("Current (A)")
plt.ylabel("Frequency (MHz)")
plt.show()

# Part (d)
print("\nPart (d)")
sigma_f = 0.2 + 0.02 * frequency
weights = 1/sigma_f**2
I_bar = np.sum(weights * current)/np.sum(weights)
f_bar = np.sum(weights * frequency)/np.sum(weights)
sigma_m = np.sqrt(np.sum(weights)/(np.sum(weights) * np.sum(weights * (current - I_bar)**2)))
```

```
sigma_b = np.sqrt(np.sum(weights * current**2)/(np.sum(weights) * np.sum(weights * (current - I_bar)**2)))

m_weight = np.sum(weights * (current - I_bar) *
                  (frequency - f_bar))/np.sum(weights * (current - I_bar)**2)
b_weight = f_bar - m * I_bar

fit_vals_weighted = m_weight * current + b_weight
chisq_weight = chisq = np.sum(( (frequency - fit_vals_weighted)/(sigma_f))**2)
print(f"Weighted chi^2: {chisq_weight}, weighted reduced chi^2 {chisq_weight/ddof}")

plt.title("Data with variable uncertainty")
plt.errorbar(current, frequency, yerr=sigma_f, fmt = "o", ls = "none", c = "purple")
plt.plot(current, fit_vals_weighted)
plt.xlabel("Current (A)")
plt.ylabel("Frequency (MHz)")
plt.show()

# Part (e)
print("\nPart (e)")
scale_factor = np.sqrt(chisq_weight/ddof)
sigma_m_adjust = sigma_m * scale_factor
sigma_b_adjust = sigma_b * scale_factor
sigma_f *= 1/scale_factor

print(f"Adjusted slope: {sigma_m_adjust}, adjusted b: {sigma_b_adjust}")

plt.title("Data with variable uncertainty, adjusted")
plt.errorbar(current, frequency, yerr=sigma_f, fmt = "o", ls = "none", c = "purple")
plt.plot(current, fit_vals_weighted)
plt.xlabel("Current (A)")
plt.ylabel("Frequency (MHz)")
plt.show()
```

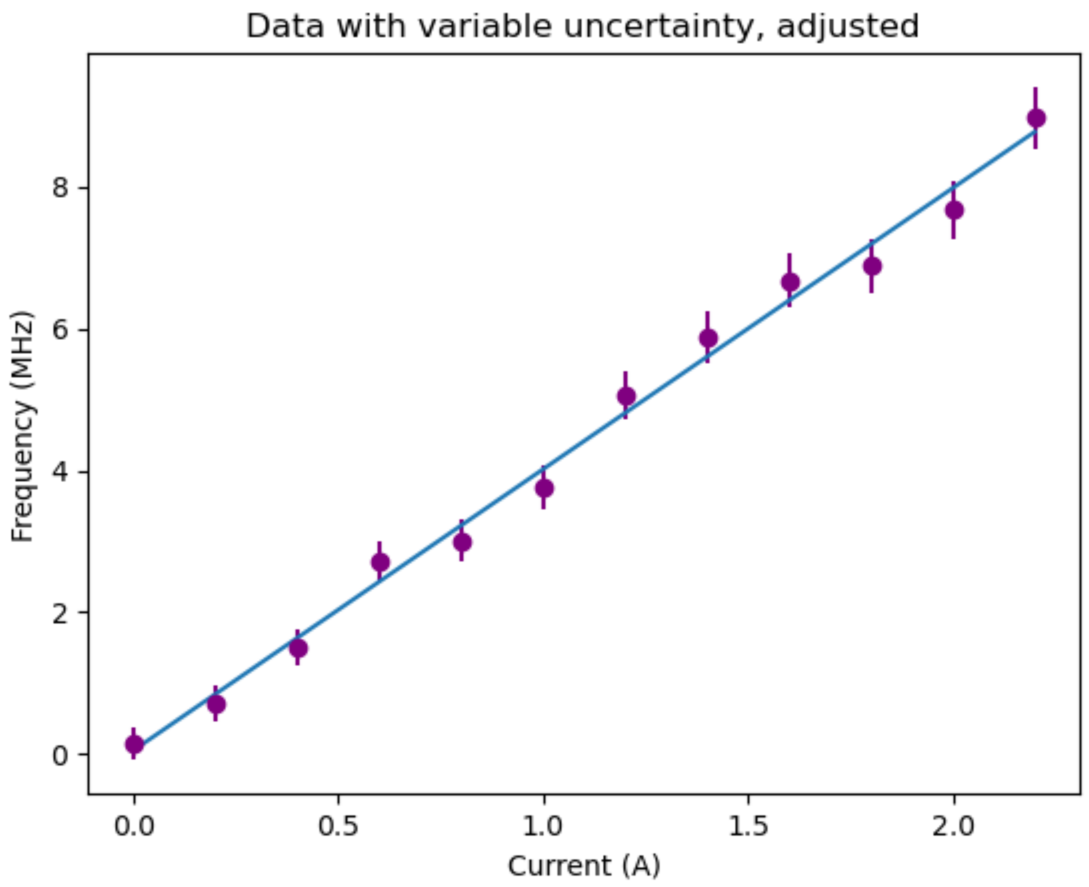
```
Part (a)
Fitted m: 3.963986013986016, b: 0.05628205128204833

Part (b)
P-value: 0.977994864918359, chi^2: 20.87184523295632, reduced chi^2: 2.0871845232956323

Part (c)
m = 3.963986013986016 ± 0.10873138338746956, b = 0.05628205128204833 ± 0.1412113303890846
```



```
Part (d)
Weighted chi^2: 7.704328145590931, weighted reduced chi^2 0.7704328145590931
```



```
Part (e)
Adjusted slope: 0.10319387330570903, adjusted b: 0.11015191022237597
```



Again, I'll write down my explanations here. There's a lot of math in this question that I will skip, and I will also largely credit Teja Nivarthi (tejanivarthi@berkeley.edu) for guiding me through this problem.

(a) The trend that we aim to show is that f is a linear plot, so we have $f = mI + b$, wehre I is the current. Then, to using least squares, we have:

$$S = \sum_i f_i - (mI_i + b)$$

and obviously our goal is to minimize this, so we take $\frac{\partial S}{\partial m}$ and $\frac{\partial S}{\partial b}$ to find the values that minimize the squared error. Simplifying this, we get the following two equations:

$$m = \frac{n\sum_i I_i f_i - \sum_i I_i \sum_i f_i}{n\sum_i I_i^2 - (\sum_i I_i)^2}$$

$$b = \frac{\sum_i f_i - m\sum_i I_i}{n}$$

We can then compute these values via python, which we do above.

(b) Following the hint, we use `scipy.stats.chi2.cdf` to determine the χ^2 value, and we get 0.977, which is larger than 0.05, meaning that in theory we have a good fit. However, looking at the plot in part (c) and our error bars, we see that the error bars don't really fit the data very well. In fact, the fit line falls outside of most error bars. Therefore, this indicates to us that the error that was hastily reported is way too small, and the probability that the measured points are where they are is actually exceedingly low.

(c) The uncertianty in the frequency is the RMS of the squared error,

$$\sigma_f = \sqrt{\frac{1}{n-2}\sum_i (f_{\text{obs},i} - f_{\text{fit},i})}$$

And the respective uncertainties in the fit parameters m and b are:

$$m = \sigma_f \sqrt{\frac{n}{n\sum_i I_i^2 - (\sum_i I_i)^2}}$$

$$\sigma_b = \sigma_f \sqrt{\frac{\sum_i I_i^2}{n\sum_i I_i^2 - (\sum_i I_i)^2}}$$

with these formulas, we can calculate σ_m and σ_b, as done in the python cell.

(d) When we introduce weighting into the equation, the equation for S changes a little bit:

$$S = \sum_i \frac{(f_i - (mI_i + b))^2}{\sigma_{f,i}^2}$$

Overall, this doesn't really change much -- all we have to do is take a weighted sum over all our data points, so our current and frequency now become:

$$\bar{I} = \frac{\sum_i w_i I_i}{\sum_i w_i} \quad \bar{f} = \frac{\sum_I w_i f_i}{\sum_i w_i}$$

With this in mind, the new formula for m is now:

$$m = \frac{\sum_i w_i (I_i - \bar{I})(f_i - \bar{f})}{\sum_i w_i (I_i - f_i)^2}$$

And with this value of m we can solve for b likewise

$$b = \bar{f} - m\bar{I}$$

Then, we use the same formula for χ^2 as we used in problem (4):

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}$$

to get a χ^2 value for this problem.

(e) Now, in order to adjust the uncertainties so that the reducded $\chi^2 = 1$, then we need to scale everything by a constant scaling factor. This scaling factor is given by $s = \sqrt{\chi^2_{\text{reduced}}}$, and the uncertainties σ_m and σ_b are multiplied by s:

$$\sigma_{m,\text{adjusted}} = \sigma_m s \quad \sigma_{b,\text{adjusted}} = \sigma_b s$$

The uncertainty in the frequency is also needs to be adjusted, here with a factor of $1/s$, so $f_{\text{adjusted}} = \frac{f}{s}$