Homework05

April 8, 2021

0.0.1 Victor Cruz Ramos.

Section 101.

1 Homework 5: Fitting

**** Submit this notebook to bourses to receive a credit for this assignment. **** Please complete this homework assignment in code cells in the iPython notebook. Please submit both a PDF of the jupyter notebook to bcourses and the notebook itself (.ipynb file). Note, that when saving as PDF you don't want to use the option with latex because it crashes, but rather the one to save it directly as a PDF.

1.1 Problem 1: Gamma-ray peak

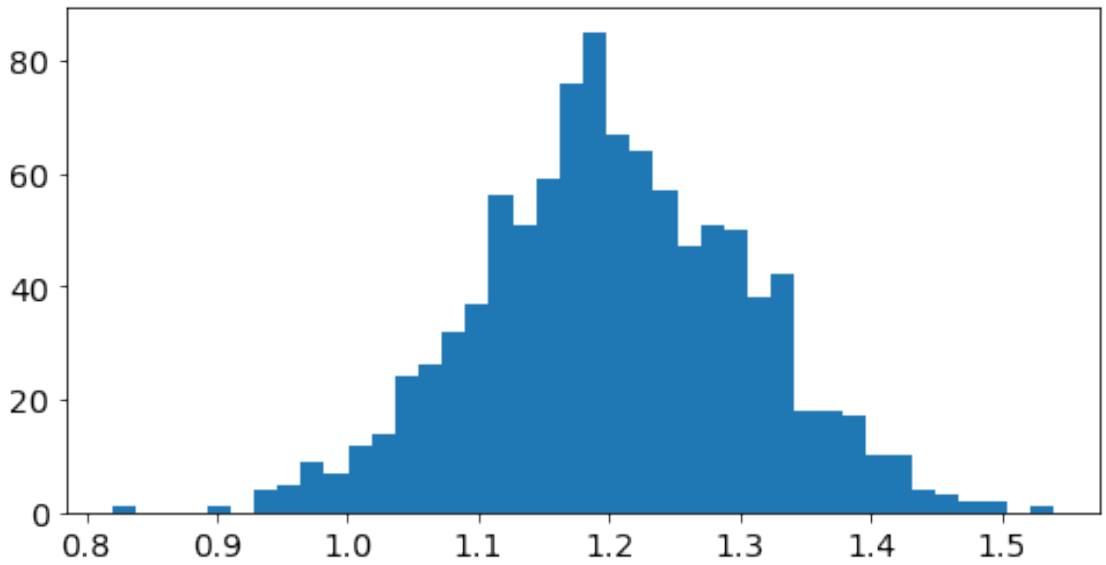
[Some of you may recognize this problem from Advanced Lab's Error Analysis Exercise. That's not an accident. You may also recognize this dataset from Homework04. That's not an accident either.]

You are given a dataset (peak.dat) from a gamma-ray experiment consisting of ~1000 hits. Each line in the file corresponds to one recorded gamma-ray event, and stores the 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: 1. 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. Since this plot represents randomly-collected data, plotting error bars would be appropriate. 1. 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 Homework04 1. Fit the 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 above. 1. Re-make your histogram from (1) with twice as many bins, and repeat the binned least-squares fit from (3) on the new histogram. How sensitive are your results to binning ? 1. How consistent is the distribution with a Gaussian? In other words, compare the histogram from (1) to the fitted curve, and compute a goodness-of-fit value, such as χ^2/d.f.

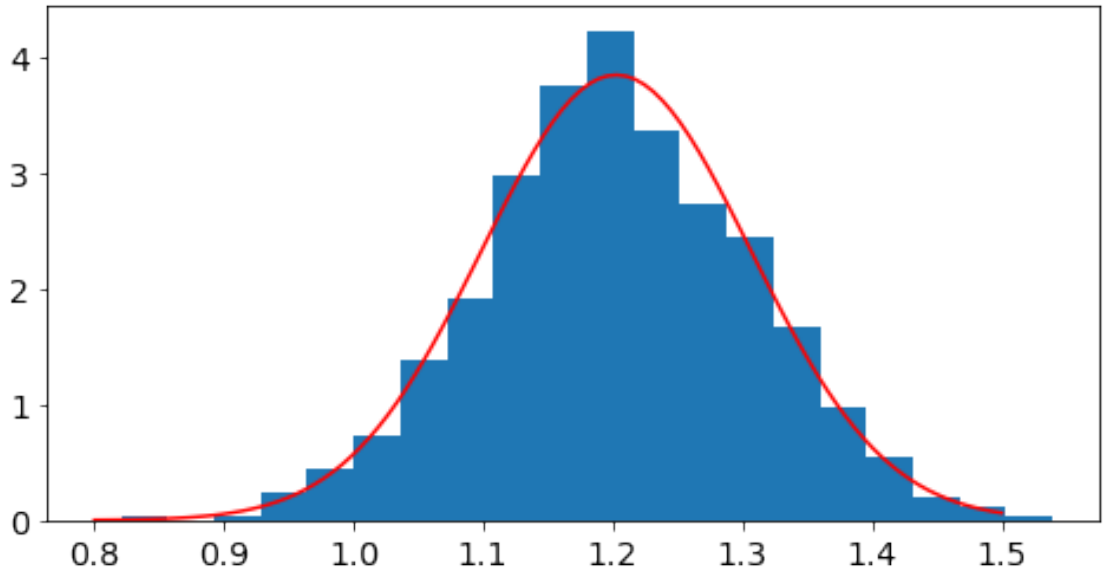
[91]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
```

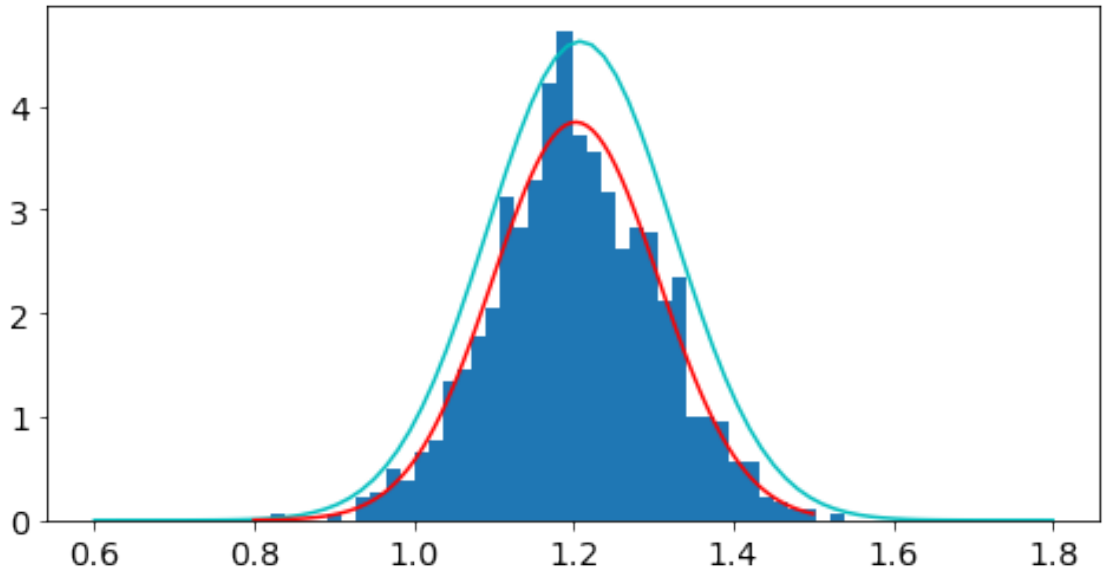
```
from scipy.stats import norm
import scipy.optimize as fitter


# Once again, feel free to play around with the matplotlib parameters
plt.rcParams['figure.figsize'] = 8,4
plt.rcParams['font.size'] = 14


energies = np.loadtxt('peak.dat') # MeV

plt.hist(energies, 40) #original bins was 20
plt.show()

x = scipy.stats.norm.fit(energies)

print(x)

x_2 = np.linspace(.8,1.5,100)
# fitted distribution
pdf_fitted = norm.pdf(x_2,loc=x[0],scale=x[1])


plt.hist(energies, 20, density=True)
plt.plot(x_2, pdf_fitted, 'r')
plt.show()


def model_newest(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))


x_values = np.linspace(.6, 1.8, 80) # the last number represents the bins;␣
 ↪original bin number was 40

xdata = np.array([0.8,1.1,1.4])
ydata = np.array([.01,3.0,1.2])
sigma = np.array([1.0,1.0,1.0])*0.2
par01 = np.array([1,1,10])
par, cov = fitter.curve_fit(model_newest, xdata, ydata, p0=par01, sigma=sigma,␣
 ↪absolute_sigma=True)
v = model_newest(x_values, par[0], par[1], par[2])
plt.plot(x_values, v, 'c')
plt.hist(energies, 40, density=True)
plt.plot(x_2, pdf_fitted, 'r') # x_values, v, 'c'#
plt.show()
```

```
print(""" 4. I noticed that the more bins there are the more accurate the␣
 ↪histogram becomes, but we also
have to be careful that we do not use too many bins because that may ruin the␣
 ↪histogram""")


""" 5. I think the histogram from 1 is consistent with a gaussian distribution␣
 ↪after looking at
some examples from lecture and other notebooks""")

#x = scipy.stats.norm.fit(energies)
#plt.hist(x, 20)
```




```
(1.202680265, 0.1037851246060088)
```





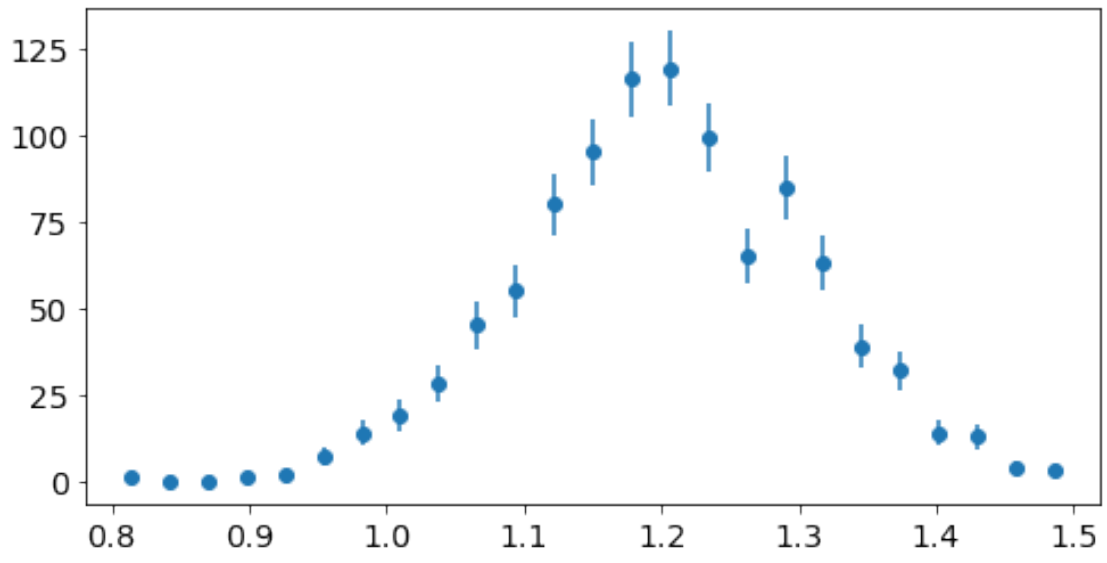
4. I noticed that the more bins there are the more accurate the histogram becomes, but we also
have to be careful that we do not use too many bins because that may ruin the histogram 5. I think the histogram from 1 is consistent with a gaussian distribution after looking at
some examples from lecture and other notebooks

Recall `plt.hist()` isn't great when you need error bars, so it's better to first use `np.histogram()` – which returns the counts in each bin, along with the edges of the bins (there are $n + 1$ edges

for n bins). Once you find the bin centers and errors on the counts, you can make the actual plot with `plt.bar()`. Start with something close to `bins = 25` as the second input parameter to `np.histogram()`.

[92]:
```
# use numpy.histogram to get the counts and bin edges

y, bin_edges = np.histogram(energies, range=(.8, 1.5),bins=25)

error_y = np.sqrt(y)

bin_centers = .5*(bin_edges[1:]+bin_edges[:-1]) # works for finding the bin␣
↪centers

# assume Poisson errors on the counts - errors go as the square root of the␣
↪count

plt.errorbar(bin_centers, y, error_y, fmt='o')

# now use plt.bar() to make the histogram with error bars (remember to label␣
↪the plot)
```

[92]:
```
<ErrorbarContainer object of 3 artists>
```



You can use the list of `energies` directly as input to `scipy.stats.norm.fit()`; the returned values are the mean and standard deviation of a fit to the data.

[93]:
```
# Find the mean and standard deviation using scipy.stats.norm.fit()
print(scipy.stats.norm.fit(energies))
'''very similar to the real values from hw 4'''
```

```
# Compare these to those computed in the previous homework (or just find them␣
↪again here)
```

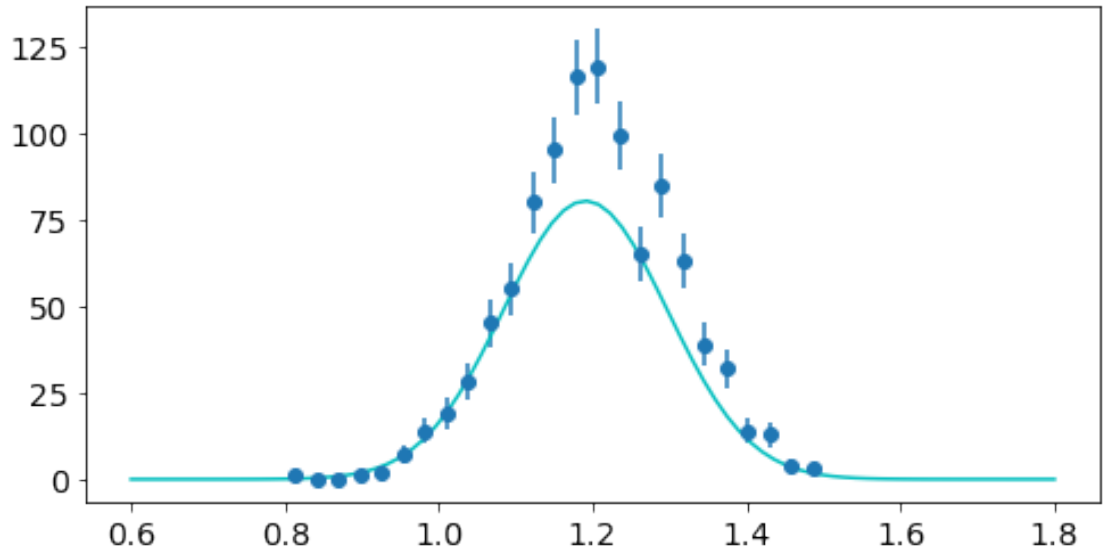
```
(1.202680265, 0.1037851246060088)
```

[93]:
```
'very similar to the real values from hw 4'
```

Now, using the binned values (found above with `np.histogram()`) and their errors use `scipy.optimize.curve_fit()` to fit the data.

[94]:
```
# Remember, curve_fit() will need a model function defined
def model(x, A, mu, sigma):
    '''Model function to use with curve_fit();
       it should take the form of a 1-D Gaussian'''
    return A*np.exp(-(x-mu)**2/(2*sigma**2))


x_values = np.linspace(.6, 1.8, 80) # the last number represents the bins;␣
↪original bin number was 40

xdata = np.array([0.8,1.2,1.5])
ydata = np.array([.1,80.0,1.2])   # I had to adjust the ydata to fit the curve␣
↪fit
sigma = np.array([1.0,1.0,1.0])*0.2
par01 = np.array([100,0,10])

par, cov = fitter.curve_fit(model_newest, xdata, ydata, p0=par01, sigma=sigma,␣
↪absolute_sigma=True)

v = model_newest(x_values, par[0], par[1], par[2])

plt.plot(x_values, v, 'c')
plt.errorbar(bin_centers, y, error_y, fmt='o')
plt.show()

# Also make sure you define some starting parameters for curve_fit (we␣
↪typically called these par0 or p0 in the past workshop)

#'''# You can use this to ensure the errors are greater than 0 to avoid␣
↪division by 0 within fitter.curve_fit()
#for i, err in enumerate(counts_err):
    #if err == 0:
        #counts_err[i] = 1

# Now use fitter.curve_fit() on the binned data and compare the best-fit␣
↪parameters to those found by scipy.stats.norm.fit()
```

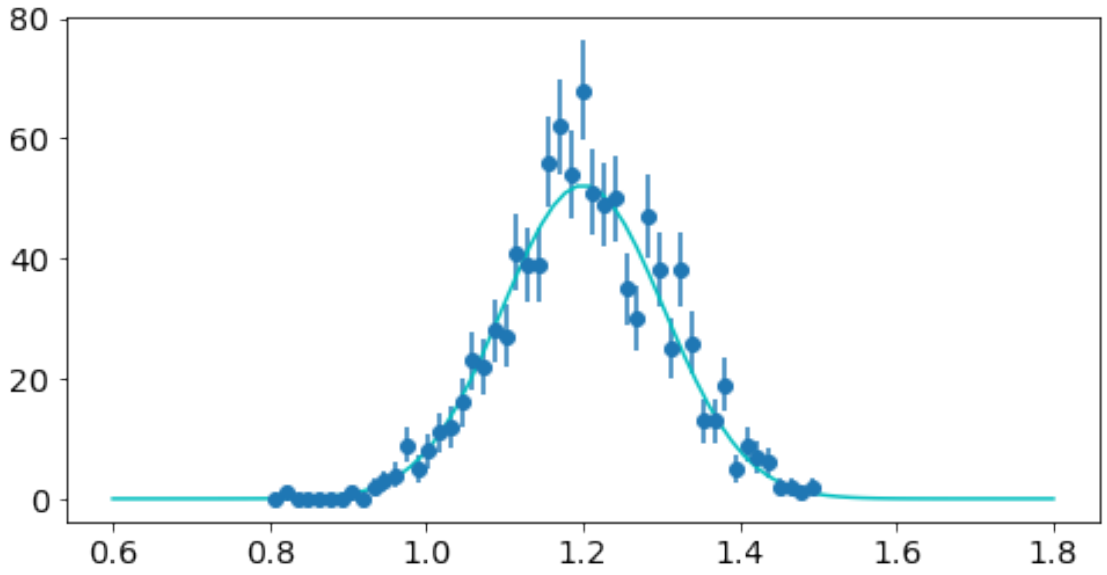
```
# It's also useful to plot the fitted curve over the histogram you made in part␣
↪1 to check that things are working properly

# At this point, it's also useful to find the chi^2 and reduced chi^2 value of␣
↪this binned fit
```



Repeat this process with twice as many bins (i.e. now use `bins = 50` in `np.histogram()`, or a similar value). Compute the χ^2 and reduced χ^2 and compare these values, along with the best-fit parameters between the two binned fits. Feel free to continue to play with the number of bins and see how it changes the fit.

[95]:
```
y, bin_edges = np.histogram(energies, range=(.8, 1.5),bins=50)

error_y = np.sqrt(y)

bin_centers = .5*(bin_edges[1:]+bin_edges[:-1])

x_values = np.linspace(.6, 1.8, 80)

y_new = []
x_new = []
error_new = []

for i in range(len(y)):
    if y[i] != 0:
        y_new.append(y[i])
        x_new.append(bin_centers[i])
        error_new.append(error_y[i])
```

```
par, cov = fitter.curve_fit(model_newest, x_new, y_new, sigma=error_new,␣
 ↪absolute_sigma=True)

v = model_newest(x_values, par[0], par[1], par[2])

plt.plot(x_values, v, 'c')
plt.errorbar(bin_centers, y, error_y, fmt='o')
plt.show()


chi_squared = np.sum(((model(x_new, *par)-y_new)/error_new)**2) # i am unsure␣
 ↪if i did the chi squared test correctly
reduced_chi_squared = (chi_squared)/(len(x_new)-len(par))
print ('chi^2 = {0:5.2f}'.format(chi_squared))
print ('reduced chi^2 = {0:5.2f}'.format(reduced_chi_squared))
```




```
chi^2 = 40.74
reduced chi^2 =  1.02
```

1.2 Problem 2: Optical Pumping experiment

One of the experiments in the 111B (111-ADV) lab is the study of the optical pumping of atomic rubidium. In that experiment, we measure the resonant frequency of a Zeeman transition as a function of the applied current (local magnetic field). 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)

0.14

0.60

1.21

1.94

2.47

3.07

3.83

4.16

4.68

5.60

6.31

6.78

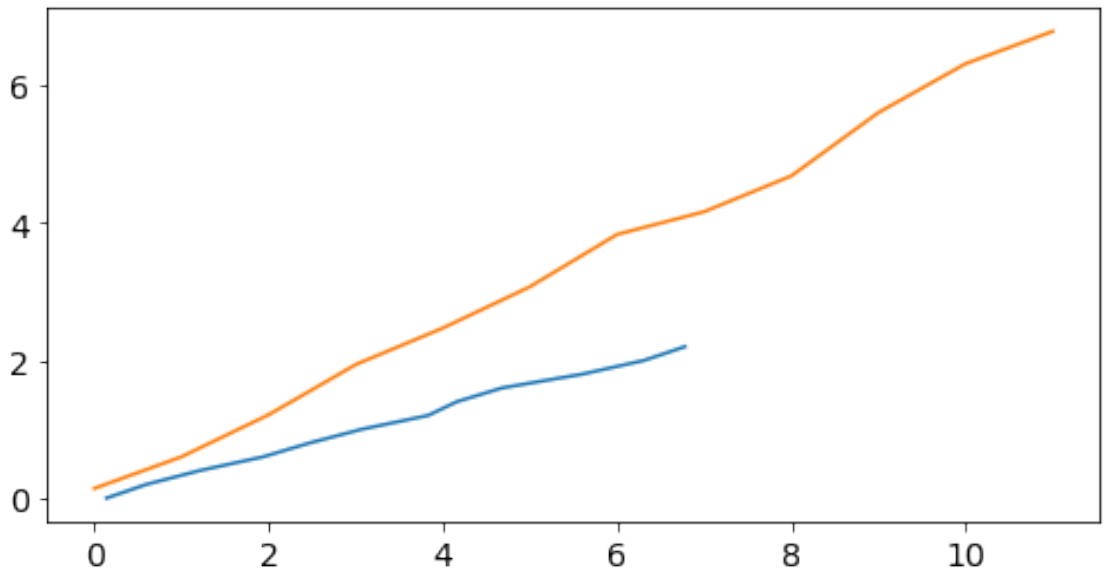
1. Plot a graph of the pairs of values. Assuming a linear relationship between I and f, determine the slope and the intercept of the best-fit line using the least-squares method with equal weights, and draw the best-fit line through the data points in the graph.
2. From what s/he knows about the equipment used to measure the resonant frequency, your lab partner hastily estimates the uncertainty in the measurement of f to be $\sigma(f) = 0.01$ MHz. Estimate the probability that the straight line you found is an adequate description of the observed data if it is distributed with the uncertainty guessed by your lab partner. (Hint: use scipy.stats.chi2 class to compute the quantile of the chi2 distribution). What can you conclude from these results?

3. Repeat the analysis assuming your partner estimated the uncertainty to be $\sigma(f) = 1$ MHz. What can you conclude from these results?
4. Assume that the best-fit line found in Part 1 is a good fit to the data. Estimate the uncertainty in measurement of y from the scatter of the observed data about this line. Again, assume that all the data points have equal weight. Use this 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.
5. Now assume that the uncertainty in each value of f grows with f: $\sigma(f) = 0.03 + 0.03 * f$ (MHz). Determine the slope and the intercept of the best-fit line using the least-squares method with unequal weights (weighted least-squares fit)

[96]:
```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import *
import scipy.stats
import scipy.optimize as fitter

# Use current as the x-variable in your plots/fitting
current = np.arange(0, 2.3, .2)  # Amps
frequency = np.array([.14, .6, 1.21, 1.94, 2.47, 3.07, 3.83, 4.16, 4.68, 5.6, 6.
↪31, 6.78]) # MHz
sigma_unequal = .03+.03*frequency
print(sigma_unequal)
plt.plot(frequency, current)
plt.plot(frequency)
```

```
[0.0342 0.048  0.0663 0.0882 0.1041 0.1221 0.1449 0.1548 0.1704 0.198
 0.2193 0.2334]
```

[96]:
```
[<matplotlib.lines.Line2D at 0x7fa430ab0790>]
```

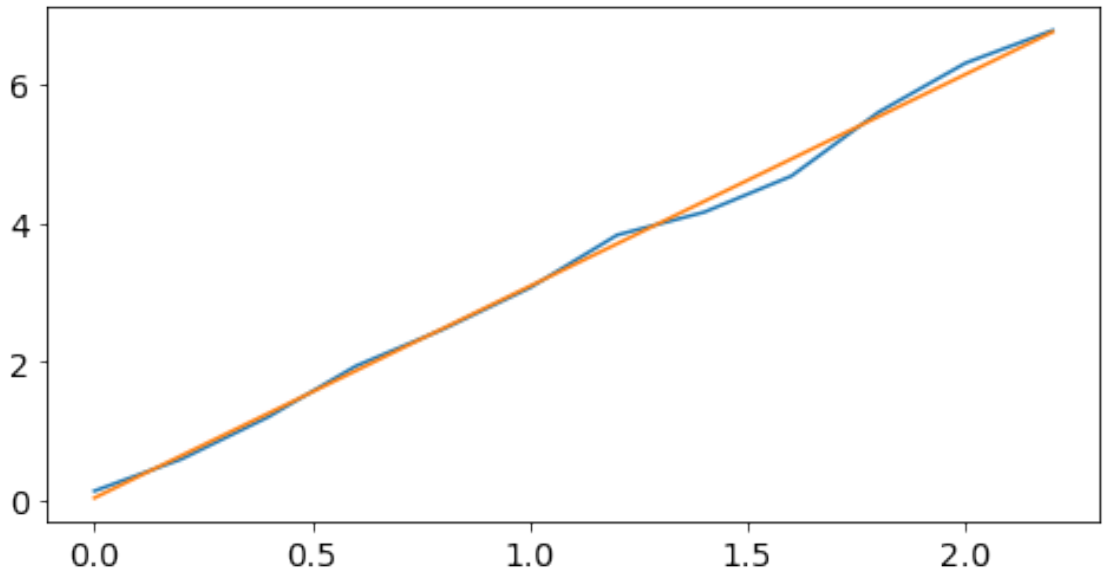


[97]:
```
linear_model = lambda x, m, b: m*x + b
sigma = 1*np.ones(len(current))

par, cov = fitter.curve_fit(linear_model, current, frequency, sigma = sigma) #␣
 ↪I recieved help from classmates who told me to use a lambda function
x = np.linspace(0, 7, 12)

plt.plot(current, frequency)
plt.plot(current, linear_model(current, *par))

expected = 1*np.ones(len(frequency))

bins = np.arange(1,7,1)
chi2 = 0
for i in range(len(current)):
    value = frequency[i]
    fit_val = linear_model(current[i], *par)
    chi2 += ((value - fit_val)/sigma[i])**2

print('chi2 = {chi2:4.6f}'.format(chi2=chi2))



scipy.stats.chi2(current)


prob = 1 - scipy.stats.chi2.cdf(chi2,len(frequency)-len(par))
print('probability = {probability:4.10f}'.format(probability = prob))

#4)

par, cov = fitter.curve_fit(linear_model, current, frequency, sigma = sigma)
print(np.sqrt(cov[0, 0]))
print('slope = {slope:4.6f} +/- {d_slope:4.6f}'.format(slope=par[0], d_slope =␣
 ↪np.sqrt(cov[0, 0]) ))
print('intercept = {b:4.6f} +/- {d_b:4.6f}'.format(b=par[1], d_b = np.
 ↪sqrt(cov[1, 1]) ))

'''3.) After finding the chi^2 for both sigma = 1 Mhz and .01 Mhz, I can␣
 ↪conclude that 1 Mhz is more accurate
since the probability is 1 and the chi^2 value is much smaller compared to .01␣
 ↪Mhz '''
```

```
chi2 = 0.151799
probability = 0.9999999803
```

```
0.05151536700545432
slope = 3.054021 +/- 0.051515
intercept = 0.039744 +/- 0.066904
```

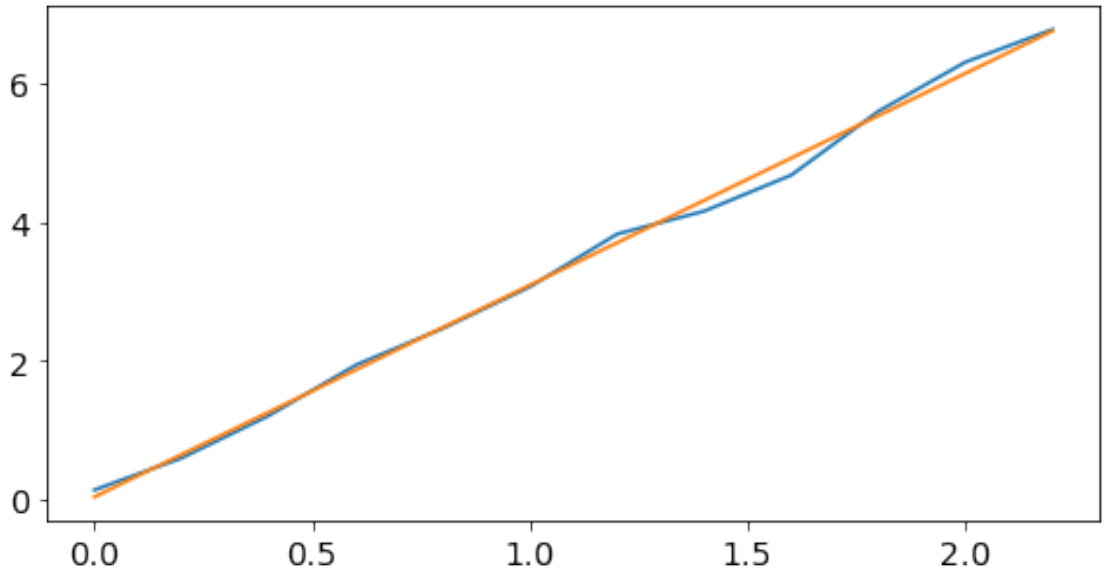
[97]:
```
'3.) After finding the chi^2 for both sigma = 1 Mhz and .01 Mhz, I can conclude
that 1 Mhz is more accurate\nsince the probability is 1 and the chi^2 value is
much smaller compared to .01 Mhz '
```



[98]:
```
sigma = .01*np.ones(len(current))

par, cov = fitter.curve_fit(linear_model, current, frequency, sigma = sigma)
x = np.linspace(0, 7, 12)

plt.plot(current, frequency)
plt.plot(current, linear_model(current, *par))

expected = 1*np.ones(len(frequency))

bins = np.arange(1,7,1)
chi2 = 0
for i in range(len(current)):
    value = frequency[i]
    fit_val = linear_model(current[i], *par)
    chi2 += ((value - fit_val)/sigma[i])**2

print('chi2 = {chi2:4.6f}'.format(chi2=chi2))
```

```
scipy.stats.chi2(current)


prob = 1 - scipy.stats.chi2.cdf(chi2,len(frequency)-len(par))
print('probability = {probability:4.10f}'.format(probability = prob))
```

```
chi2 = 1517.991841
probability = 0.0000000000
```



[99]:
```
#5.
par, cov = fitter.curve_fit(linear_model, current, frequency, sigma =␣
 ↪sigma_unequal)
print(np.sqrt(cov[0, 0]))
print('slope = {slope:4.6f} +/- {d_slope:4.6f}'.format(slope=par[0], d_slope =␣
 ↪np.sqrt(cov[0, 0]) ))
print('intercept = {b:4.6f} +/- {d_b:4.6f}'.format(b=par[1], d_b = np.
 ↪sqrt(cov[1, 1]) ))
```

```
0.050215609233483276
slope = 2.990539 +/- 0.050216
intercept = 0.089167 +/- 0.030170
```

The rest is pretty short, but the statistics might be a bit complicated. Ask questions if you need advice or help. Next, the problem is basically asking you to compute the χ^2 for the above fit twice, once with 0.01 as the error for each point (in the 'denominator' of the χ^2 formula) and once with 1.

These values can then be compared to a "range of acceptable χ^2 values", found with

`scipy.stats.chi2.ppf()` – which takes two inputs. The second input should be the number of degrees of freedom used during fitting (# data points minus the 2 free parameters). The first input should be something like 0.05 and 0.95 (one function call of `scipy.stats.chi2.ppf()` for each endpoint fo the acceptable range). If the calculated χ^2 statistic falls within this range, then the assumed uncertainty is reasonable.

[100]:
```
ndf = len(frequency)-len(par) # this is for 1 Mhz
print(scipy.stats.chi2.ppf(1-0.9999999803, ndf))

# this is for .01 Mhz
print(scipy.stats.chi2.ppf(1-0.000000, ndf))
```

```
0.15179170189462318
inf
```

Now, estimate the uncertainty in the frequency measurements, and use this to find the uncertainty in the best-fit parameters. This document is a good resource for learning to propagate errors in the context of linear fitting.

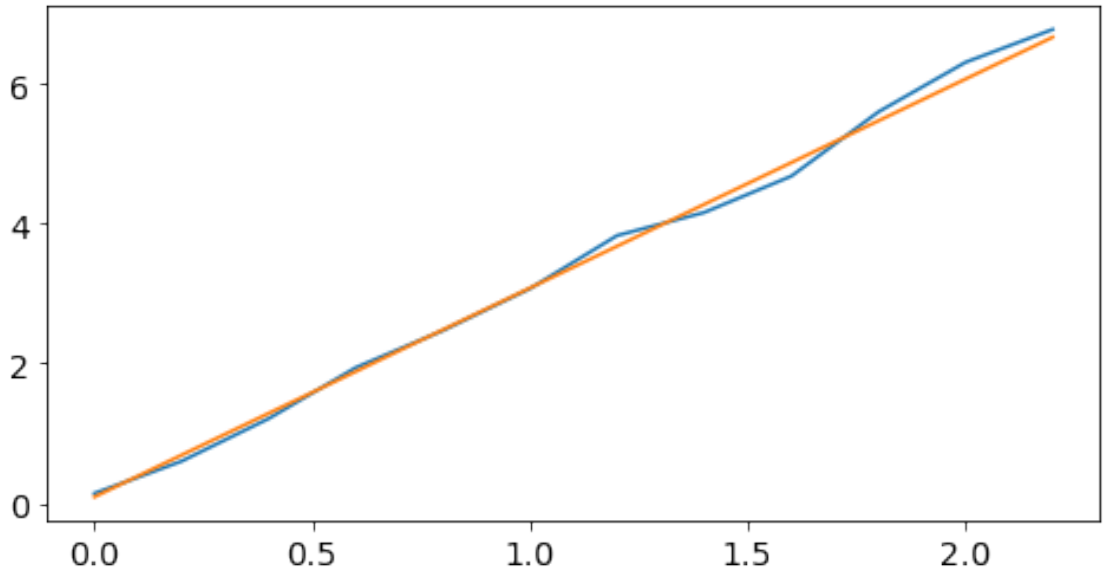
[104]:
```
datas = scipy.stats.chi2.ppf(.5, ndf)
print(datas)
```

```
9.34181776559197
```

Finally, repeat the fitting with the weighted errors (from the $\sigma(f)$ uncertainty formula) given to `scipy.optimize.curve_fit()`

[102]:
```
uncertainty  = 0
for i in range(len(current)):
    value = frequency[i]
    fit_val = linear_model(current[i], *par)
    uncertainty  += ((value - fit_val)/datas)**2 # this for loop generates the␣
 ↪new uncertainty

newest_sigma = uncertainty *np.ones(len(current))
print(uncertainty)
scipy.optimize.curve_fit(linear_model, current, frequency, sigma = newest_sigma)

plt.plot(current, frequency)
plt.plot(current, linear_model(current, *par))

chi2 = 0
for i in range(len(current)):
    value = frequency[i]
    fit_val = linear_model(current[i], *par)
    chi2 += ((value - fit_val)/sigma[i])**2
print('chi2 = {chi2:4.6f}'.format(chi2=chi2)) # I think this fit is all that is␣
 ↪required; I'm quite sure this all we had to do . . .
```

```
0.0020608386230391872
chi2 = 1798.484781
```



[ ]: