ENGP 3170 Homework 3

Matt Catalano and Ian Pimenta

March 3, 2021

1 Engineering Problem Statement

Comprehensive engineering analysis is dependent on interpreting and modeling data. In this assignment, we got hands on experience by modeling and analyzing data from a π meson decay experiment. By plotting a linear fit line over this experimental data, we were able to determine the lifetime, τ, while also interpreting the correlation of the parameters used for our model. Manipulating our σ_i allowed us to make our error bars more realistic. We also got practice fitting nonlinear functions to our lagrange energy/cross-section data and analyzing fit using chi squared analysis.

2 Numerical Methods

Chi-Squared: $\chi^2 = \frac{(y_i - f(a,b,c))^2}{\sigma_i^2}$

Standard Model of Decay: $n(t) = n(0)e^{-\lambda t}$

Percent error: $\%\ error = \frac{|tabulated\ value - experimental|}{tabulated\ value} \times 100\%$

Degrees of Freedom: $Number\ of\ Data\ Points\ -\ Number\ of\ Parameters$

Reduced Chi-Squared: $\frac{\chi^2}{Degrees of Freedom}$

Fit Function (Problem 4): $y = \frac{A}{(1+B(x-C)^2)}$

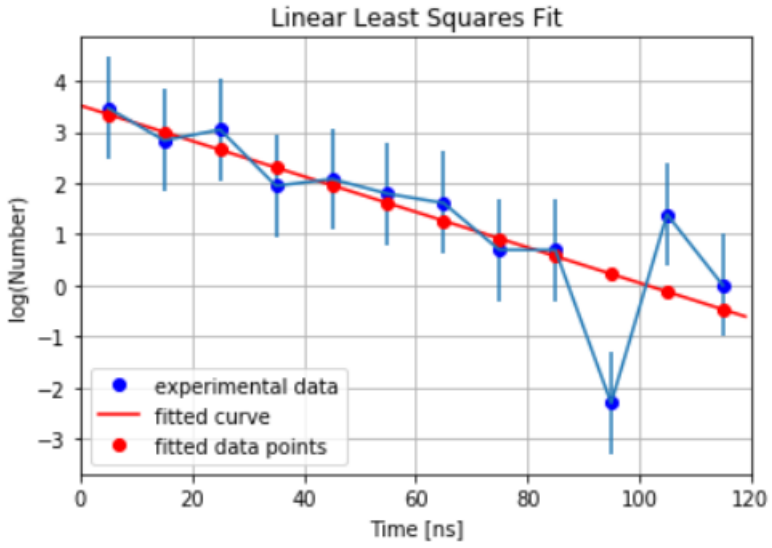
Equation to fit (Problem 5): $y = \frac{A}{(D+B(x-C)^2)}$

3 Question 1

Use the code provided to determine the lifetime τ of the π meson. Compare to the tabulated lifetime of 2.6×10^{-8} seconds, and comment on the difference. Do the two parameters (intercept and slope) turn out to be independent, or is there a strong correlation between them? If there is a correlation, does the sign of the correlation make sense? Make sure that you provide appropriate x and y axis labels in the figure (indicating quantities plotted and units), along with a key to show what is being plotted (original data points and fit), and an informative caption.

3.1 Code and Work

```
""" From "COMPUTATIONAL PHYSICS", 3rd Ed, Enlarged Python eTextBook
    by RH Landau, MJ Paez, and CC Bordeianu
    Copyright Wiley-VCH Verlag GmbH & Co. KGaA, Berlin;  Copyright R Landau,
    Oregon State Unv, MJ Paez, Univ Antioquia, C Bordeianu, Univ Bucharest, 2015.
    Support by National Science Foundation

    adapted by Lev Kaplan in 2019 for linear decay fit y(x) = a + b x """

from pylab  import*

x = range(5,120,10)    # time from 5 to 115 in steps of 10 (12 points)
Nd = len(x)   # number of data points
y = log([32,17,21,7,8,6,5,2,2,0.1,4,1])   # log of number of counts
sig = [1] * 12   # error bars all set to 1

plot(x, y, 'bo', label = 'experimental data' )                                  # Plot data in blue

errorbar(x,y,sig)                                       # Plot error bars
title('Linear Least Squares Fit')                          # Plot figure
xlabel( 'Time [ns]' )                                             # Label axes
ylabel( 'log(Number)' )
grid(True)                                                 # plot grid
xlim(0,120)                                                # x range for plot

ss = sx = sxx = sy = sxy = 0   # initialize various sums

for i in range(0, Nd):         # compute various sums over data points
    sig2 = sig[i] * sig[i]
    ss += 1. / sig2;    sx += x[i]/sig2;        sy += y[i]/sig2
    sxx += x[i] * x[i]/sig2;    sxy += x[i]*y[i]/sig2;

delta = ss*sxx-sx*sx
slope = (ss*sxy-sx*sy) / delta    #calculate best fit slope
inter = (sxx*sy-sx*sxy) / delta   # calculate best fit intercept

print('Linear Fit Final Results\n')
print('y(x) = a + b x')                                # Desired fit
print('a = ', inter, '+/-', sqrt(sxx/delta))
print('b = ', slope, '+/-', sqrt(ss/delta))
print('correlation =',-sx/sqrt(sxx*ss))

# red line is the fit, red dots the fits at y[i]
t = range(0,120,1)
curve  = inter + slope*t
points = inter + slope*x
plot(t, curve,'r', label='fitted curve')
plot(x, points, 'ro', label='fitted data points')
legend(loc = 'lower left')
show()
```



Demonstration of linear least squares fit obtained from pi meson decay experiment

```
tau = -1/slope
print('Tau = ' + str(tau) + ' ns')
```

```
Tau = 28.83641884753647 ns
```

3.2 Input and Parameters

The number of counts at certain times is given to us as experimental data to perform a linear least squares fit on. Using this data, a change of variables to set the y axis as log(counts), and setting all error bars equal to 1, linear least squares fit is performed.

3.3 Results

To find the lifetime, we first need to understand the decay model and what happens when we use our logarithmic change of variables to this model. The standard decay model appears like so:

$$n(t) = n(0)e^{-\lambda t}$$

In this format, the lifetime, or τ is equal to $\frac{1}{\lambda}$. When we perform the logarithmic change of variables, the formula changes to appear like this:

$$y = y(0) - \lambda t$$

Therefore, when looking at our final, logarithmically changed function, we can say that $\lambda = -slope$. We then get a value of tau roughly equal to 2.8836×10^{-8} s. When we compare our value to the tabulated lifetime, 2.6×10^{-8}, we can see that they only differ by ≈ 2.8ns. Using the percent error formula referenced in our numerical methods section, we found the % error:

$\% \; error = \frac{|tabulated\ value - experimental|}{tabulated\ value} \times 100\%$

$= \frac{|((2.6 \cdot 10^{-8}) - (2.8836 \cdot 10^{-8}))|}{2.6 \cdot 10^{-8}} \times 100\%$

$= 10.907\%$

$\approx 11\%$ is a non-negligible amount of error; however, contextualized within our nanosecond scale, the error is not wildly inaccurate.

```
Linear Fit Final Results

y(x) = a + b x
a =  3.517370894128999 +/- 0.5788623196249563
b =  -0.034678369921285535 +/- 0.008362420100070907
correlation = -0.8667781422175382
```

The results of the Linear Fit results show us that the correlation is roughly -0.867. Because our value is close to -1, we can say that there is a strong correlation between the two parameters (intercept and slope). The negative sign denotes negative correlation, which makes sense because this means that the intercept and slope are inversely proportional. As the intercept increases (initial count), the slope will decrease (become more negative) because the slope will always hit zero at the same point (at the end of the pi meson lifetime).

4 Question 2

Replace experimental uncertainties $\sigma_i = 1.0$ for each data point with more realistic values.

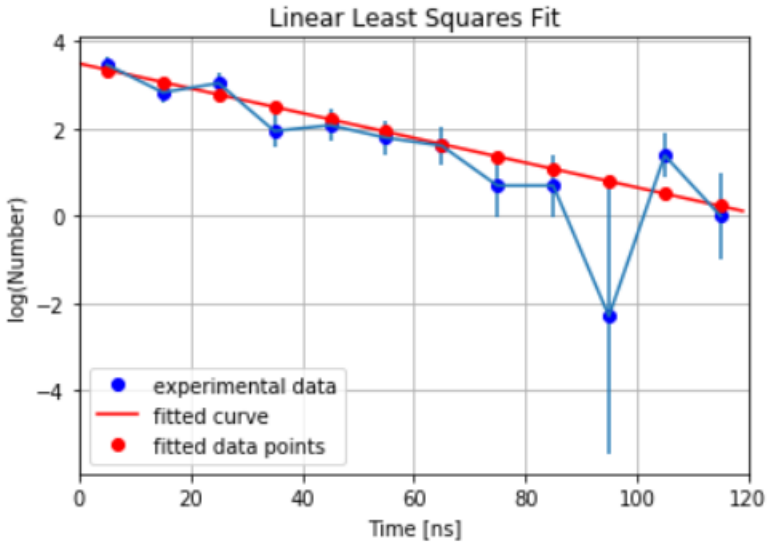
Hint: if the number of events in a given time interval is ni and we assume the counting can be described by a Poisson process, then the uncertainty (standard deviation) in n_i is $\Delta n_i = n_i^{1/2}$, as will be

discussed in class. (Strictly speaking Poisson statistics can be approximated by Gaussian statistics, justifying the least squares technique, only when the counts n_i are all large, but we're pretending that the least squares method is valid even though some n_i are in fact small.) However, we need to transform to the new variables y_i=log(n_i). Let Δy_i be a small change in y_i corresponding to a small change Δn_i. Treating Δy_i and Δn_i as infinitesimals (which again is not fully justified because n_i are not necessarily large), find $\Delta y_i/\Delta n_i$, and thus obtain Δy_i for each i. Set $\sigma_i = \Delta y_i$, which is the uncertainty in each y_i.

Re-do the fit with the new uncertainties σ_i. Plot the new fit just as in problem (1). Did the calculated values of the intercept and slope change? What is your new value for the lifetime, and what is the uncertainty in the lifetime? Find the value of χ^2 using your new best fit, by summing $\frac{(y_i-(a+bx_i))^2}{\sigma_i^2}$ over all i from 0 to Nd-1 and comment on whether the fit is reasonable. Why or why not?

4.1 Code and Work

```
import numpy as np
x = range(5,120,10)    # time from 5 to 115 in steps of 10 (12 points)
Nd = len(x)   # number of data points
n_i = [32,17,21,7,8,6,5,2,2,0.1,4,1]
n_i = np.array(n_i)

y = log(n_i)   # log of number of counts

sig = n_i**-0.5 #error bars set more realistically

plot(x, y, 'bo', label = 'experimental data' )                              # Plot data in blue

errorbar(x,y,sig)                                  # Plot error bars
title('Linear Least Squares Fit')                     # Plot figure
xlabel( 'Time [ns]' )                                         # Label axes
ylabel( 'log(Number)' )
grid(True)                                            # plot grid
xlim(0,120)                                           # x range for plot

ss = sx = sxx = sy = sxy = 0   # initialize various sums

for i in range(0, Nd):         # compute various sums over data points
    sig2 = sig[i] * sig[i]
    ss += 1. / sig2;    sx += x[i]/sig2;        sy += y[i]/sig2
    sxx += x[i] * x[i]/sig2;    sxy += x[i]*y[i]/sig2;

delta = ss*sxx-sx*sx
slope = (ss*sxy-sx*sy) / delta    #calculate best fit slope
inter = (sxx*sy-sx*sxy) / delta   # calculate best fit intercept

print('Linear Fit Final Results\n')
print('y(x) = a + b x')                           # Desired fit
print('intercept = ', inter, '+/-', sqrt(sxx/delta))
print('slope = ', slope, '+/-', sqrt(ss/delta))
print('correlation =',-sx/sqrt(sxx*ss))

# red line is the fit, red dots the fits at y[i]
t = range(0,120,1)
curve  = inter + slope*t
points = inter + slope*x
plot(t, curve,'r', label='fitted curve')
plot(x, points, 'ro', label='fitted data points')
legend(loc = 'lower left')
show()
```




```
Linear Fit Final Results

y(x) = a + b x
intercept =  3.494451334553311 +/- 0.14368984243070387
slope =  -0.028429280769792613 +/- 0.003618512998226424
correlation = -0.7342787807954047
```

```
tau = -1/slope
tau_uncertainty = (1/(slope**2))*(sqrt(ss/delta))
print('Tau = ' + str(tau) + ' +/- ' + str(tau_uncertainty))
```

```
Tau = 35.1750017208506 +/- 4.477116462080065
```

```
chi_2 = np.sum((y - (inter + slope * x))**2/(sig)**2)
chi_2
```

```
10.469097733132594
```

```
import scipy.special
parameters = 2
alpha = (Nd - parameters)/2
prob = scipy.special.gammaincc(alpha, chi_2)
prob
```

```
]: 0.021528678882132883
```

Deriving a new σ_i:

$y_i = log(n_i)$
$\Delta n_i = n_i^{0.5}$

Let Δy_i be a small change in y_i corresponding to a small change Δn_i. Therefore, we may (not fully justified) treat Δy_i and Δn_i as infinitesimals

$d(y_i) = d(log(n_i))$
$dy_i = \frac{1}{n_i} dn_i$
$dy_i = (1/n_i) * (n_i)^0.5$
$\Delta y_i = (n_i)^{-0.5}$
$\Delta y_i = \sigma_i$

Deriving uncertainty in lifetime:
$\tau = \frac{-1}{b}$
$Uncertainty = \frac{d\tau}{db} = \frac{d\frac{-1}{b}}{db} = \frac{1}{b^2}$
Therefore: $d\tau = \frac{1}{b^2} \times db$
(from code): db = sqrt(ss/delta)

4.2 Input and Parameters

Here, we used the same inputs as question #1; however, we redefined our σ_i such that it is equal to $\Delta y_i = (n_i)^{-0.5}$ (Derivation above).

4.3 Results

The intercept and slope did, in fact, change between problem 1 and problem 2. The intercept decreased slightly (originally: 3.517, now: 3.494) and the slope became less negative (originally:-0.034678, now:-0.28429). Our new τ value (with uncertainty)= 35.1750017208506 +/- 4.477116462080065. Our χ^2 value is 10.469097733132594. After performing analysis on this chi squared value (using the scipy.special.gammaincc() function), we found that the probability given by the regularized incomplete gamma function is 0.0215. This means that there is a 2.15% chance that if the experiment were performed again with random errors, the

resultant chi squared value would be larger than what we have found. Therefore, there is a 97.85% chance that the chi squared value found by random chance would be smaller (and a 97.85% chance that the fit will be better). Because of how small the probability given by the function is, we can conclude that this is a poor fit.

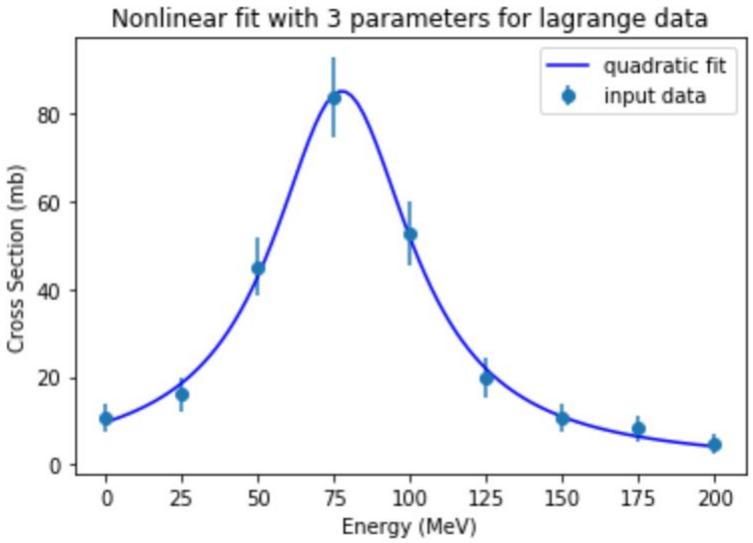
5 Question 3

The code Nonlinfit.py uses scipy.optimize.curve_fit() to perform a general fit, where the functional form may be nonlinear in the parameters. Adapt Nonlinfit.py to fit the data file lagrange.dat (provided earlier with assignment 2) to the functional form $y = \frac{A}{1+B(x-C)^2}$. Note that in the sample data file sample.txt, the uncertainties σ_i are provided in the input file (3rd column). Here you will instead use uncertainties $\sigma_i = ky_i^{1/2}$, where k is some undetermined constant (since the data points y are cross sections, which are not counts, but are proportional to counts). You may set k=1.0 initially. Don't forget that you may need to set reasonable initial guesses for A, B, C in order for the fit to converge to the right answer (see the p0=... option when calling curve_fit). Find the best values for A, B, C. What is the physical meaning of each parameter? Can you understand the signs of the correlations (or covariance matrix elements) between the parameters? Plot the best-fit function along with the original data.

5.1 Code and Work

```
""" Example of curve fitting using scipy.optimize package
    Performs chi squared fit of data insample.txt file to qudratic function

    Lev Kaplan 2019"""

# Nonlinfit.py: Langrange interpolation tabulated data

from pylab import *
from scipy.optimize import curve_fit   # chi squared fitting

def func(x,a,b,c):   # define functional form
    return a/(1+b*(x-c)**2)

NMAX = 1000  # max number of input points


k = 1

inputfile = open("lagrange.dat","r")  # read in the input x,y values
r = inputfile.readlines()  # read the whole file into list (one item per line)
inputfile.close()
        # input has the form: x0 y0 sig0
        #                     x1 y1 sig1
        #                     ...

xin = zeros(len(r))  # each is array of length NMAX, all elements set to zero
yin = zeros(len(r))
sig = zeros(len(r))

m = 0
for line in r:
    print(line)
    s = line.split() # split line and split into list of items(assume items separated by spaces)
    xin[m] = s[0] # first number in each line is the x value
    yin[m] = s[1]
    sig[m] = k * (float(s[1])**0.5)
    m+=1          # m is total number of input data points
                  # will be stored in xin[0]..xin[n-1],yin[0]..yin[n-1]

popt,pcov = curve_fit(func, xin[0:m], yin[0:m], p0=[5, 7, 9], sigma=sig[0:m])

print("best fit parameters a,b,c =",popt)

print("covariance matrix for the parameters a,b,c =\n",pcov)

print("uncertainties in parameters =",sqrt(diag(pcov)))

xvalues = linspace(0,200,100)
yvalues = func(xvalues,popt[0],popt[1],popt[2])

errorbar(xin[0:m],yin[0:m],sig[0:m],fmt="o",label="input data")
plot(xvalues,yvalues,"b-",label="quadratic fit")
legend(loc="upper right")
show()
```

```
best fit parameters a,b,c = [8.50351900e+01 1.29296779e-03 7.76567936e+01]
covariance matrix for the parameters a,b,c =
 [[1.57801760e+01 3.62087652e-04 2.82438804e-01]
 [3.62087652e-04 1.36492580e-08 1.17080812e-05]
 [2.82438804e-01 1.17080812e-05 1.23053948e+00]]
uncertainties in parameters = [3.97242697e+00 1.16830039e-04 1.10929684e+00]
```



5.2 Input and Parameters

0	10.6
25	16
50	45
75	83.5
100	52.8
125	19.9
150	10.8
175	8.25
200	4.7

The x and y values shown above comprise the lagrange.dat file used as input for our function. The functional form of our function is $y = \frac{A}{1+B(x-C)^2}$. Using this input data and error values found by setting $\sigma_i = ky_i^{1/2}$ and $k = 1$, nonlinear fit is performed.

5.3 Results

As seen in the figure above, our best fit parameters for a, b, c are: [8.50351900e+01 1.29296779e-03 7.76567936e+01]. In order to determine the meaning of each of our parameters, we can compare our function to the Lorentzian equation:

$Y = \frac{Amplitude}{(1+((X-Center)/Width)^2)^2}$

$A = amplitude$

$C = peak\ centroid\ (resonance\ energy\ or\ resonance\ frequency)$

$B = (1/width)^2.\ B,\ therefore,\ is\ the\ decay\ rate\ of\ inverse\ lifetime$

The signs of the correlations (covariance matrix elements) between the parameters indicate whether or not the parameters are negatively or positively correlated with each other. All of our covariance matrix elements are positive which means that the parameters are all positively correlated with each other. Amplitude increases as the peak increases which makes sense intuitively. Amplitude also increases as width increases as does the peak centroid. The best-fit function along with original data is plotted above.

6 Question 4

In Problem (3), obtain the value of χ^2 by summing $\frac{(y_i - f(x_i, A, B, C))^2}{\sigma_i^2}$ over all i from 0 to m-1. What is the number of degrees of freedom? Is the fit reasonable? If not, how might you adjust the uncertainties (specifically the constant k) to make the fit reasonable? What are the resulting uncertainties in the parameters A, B, C?

6.1 Code and Work

```
In [139]: chi_2 = np.sum((yin - func(xin, popt[0],popt[1],popt[2]))**2/(sig)**2)
          print('chi squared value = ' + str(chi_2))

          chi squared value = 1.2821240335057018

In [140]: Nd = 9
          parameters = 3
          alpha = (Nd - parameters)/2
          prob = (scipy.special.gammaincc(alpha, chi_2))
          prob

Out[140]: 0.86120912569055
```

```
print("uncertainties in parameters a, b, c =",sqrt(diag(pcov)))

uncertainties in parameters a, b, c = [3.97242697e+00 1.16830039e-04 1.10929684e+00]
```

6.2 Input and Parameters

Data gathered from problem 3, specifically the general fit formula, the best fit values for the parameters, and the error, is used here to find the chi squared value. Number of data points and number of parameters are then used to find the output of the scipy.special.gammaincc() function.

6.3 Results

Our chi squared value is 1.282. The number of degrees of freedom can be found by subtracting the number of parameters from the number of data points. For this problem, there were 9 data points and 3 parameters. This makes for 6 degrees of freedom. We use the scipy.special.gammaincc() function to find the probability that repeating the experiment with random error could give us a larger chi squared value. The probability found is 0.8612. This means that there is a 86.12 percent chance that if the experiment were performed

again with random errors, the resultant chi squared value would be larger than what we have found. There is only a 13.88 percent chance to find a better fit. Due to this, we can confirm that this is a reasonable fit.

The fit is already reasonable, but if it were unreasonable and we wanted to make it reasonable, we would want to increase the k proportionality constant for the error to make each error bar proportionally larger. These larger error bars would allow for higher likelihood that our curve would fit into the error boundaries. This means that it will become less likely that a similar experiment with the same random error would be able to find a smaller chi squared value, which implies that the fit became more reasonable. We need to be careful with this, however. If we increase k too much, the fit could be too good and we could be overfitting. Too large of an increase in k would make the error unrealistic to the point that any fit would be good since the error is so high.

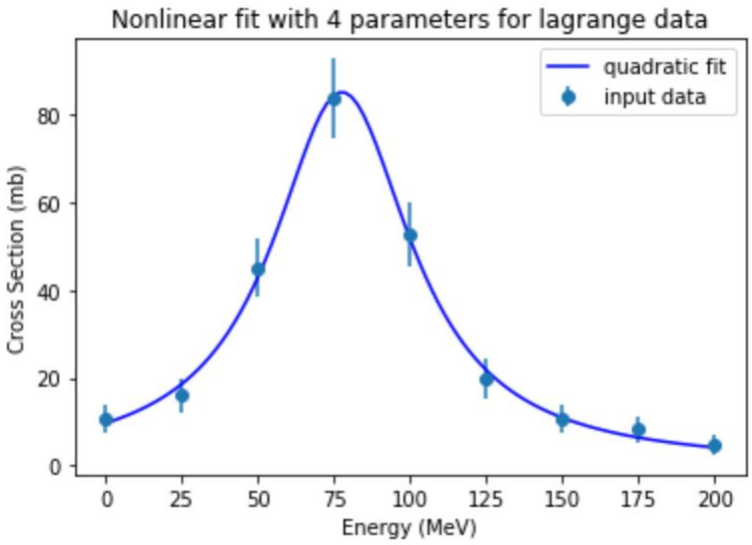
The resulting uncertainties in the parameters A, B, and C are [3.97242697e+00, 1.16830039e-04, 1.10929684e+00]. The uncertainty is highest in parameter A (amplitude) and by far the lowest in parameter B (decay rate of inverse lifetime). This tells us that we are most unsure about the correct value for amplitude but much more confident in our result for inverse lifetime decay rate.

7 Question 5

[Required for 6170 students only, extra credit for 3170 students] Generalize the functional form to $y = \frac{A}{(D+B(x-C)^2)}$, and fit for A, B, C, D. Does the fit improve? Explain the result.

7.1 Code and Work

```
New function = a/(d+b*(x-c)**2)

p0 = [5, 7, 9, 7]

best fit parameters a,b,c,d = [1.17204456e+04 1.78210434e-01 7.7656793
6e+01 1.37830540e+02]
covariance matrix for the parameters a,b,c,d =
 [[8.35351849e+20 1.27015935e+16 2.55854536e+10 9.82360836e+18]
 [1.27015935e+16 1.93128773e+11 3.89028926e+05 1.49368772e+14]
 [2.55854536e+10 3.89028926e+05 2.26028984e+00 3.00880972e+08]
 [9.82360836e+18 1.49368772e+14 3.00880972e+08 1.15524113e+17]]
uncertainties in parameters a,b,c,d = [2.89024540e+10 4.39464188e+05
1.50342603e+00 3.39888383e+08]
```




```
In [143]: chi_2 = np.sum((yin - func(xin, popt[0],popt[1],popt[2],popt[3]))**2/(sig)**2)
          chi_2

Out[143]: 1.2821240335060529

In [144]: Nd = 9
          parameters = 4
          alpha = (Nd - parameters)/2
          scipy.special.gammaincc(alpha, chi_2)

Out[144]: 0.7667896193767493
```

7.2 Input and Parameters

Input for this problem is similar to problem 4. There is a new general fit function, $y = \frac{A}{(D+B(x-C)^2)}$, and due to this there are new best fit parameters and errors. The k value is kept at k=1. This is used to find our chi squared value. Analysis of the chi squared value is performed by finding the degrees of freedom from number of parameters and number of data points and utilization of the scipy.special.gammaincc() function.

7.3 Results

When generalizing the functional form to include a fourth parameter, the chi squared value is found to be the same as the non-generalized form. Because the chi squared value remained the same, but the degrees of freedom decreased, the fit is analytically worse. When more parameters are used, it should theoretically be easier to find the best fit function, which would result in an improved chi squared value. Therefore, if the chi squared value remains the same, despite an increase in the number of parameters, the solution that has more parameters has a worst fit with respect to its potential for best fit.

8 Conclusion

In this lab, we worked to fit data linearly and non-linearly. Both fits were analyzed using chi squared analysis. In order to perform the linear fit, we had to manipulate the data into logarithm form. Using π

meson experiment data, the lifetime τ of the π meson was found experimentally through linear least squares fit and compared to the tabulated lifetime. Experimental data turned out to be very close to the tabulated lifetime. We experimented with improving error bars by changing our $\sigma_i = 1$ to $\sigma_i = \Delta y_i = (n_i)^{-0.5}$. This helped improve the fit of our error bars. A χ^2 test was also used to evaluate our data. Our linear fit data was found to be a poor fit from analysis of the χ^2 test. The nonlinear fit χ^2 analysis, however, was more promising. Nonlinear fit was used on the lagrange data received in an earlier lab. A Lorentzian style function was used for the fit and both the generalized and non-generalized Lorentzian functions produced good fits to the data. However, it was seen that generalizing the Lorentzian functional form actually made the fit less reasonable than the non-generalized form. This is because the χ^2 value remained the same for both versions while the degrees of freedom decreased.