More Advanced Model Fitting and Plotting

PHY2004W **KDSMIL001**

24 Feb 2020

Contents

1 Answers

The first section of the activity was plotting the best-fit curve for a set of non-linear points. The code for everything in this section can be found in Appendix 1. Firstly, we were asked to plot the data supplied to us in DampedData.txt, which was a text file containing time and position data of a damped oscillator. To do this we used the errorbar function in `matplotlib.pyplot` [line 31]. Next, we defined a function that takes in a set of parameters and returns a value for $y(t)$ where

$$y(t) = A + Be^{-\gamma t}cos(\omega t - \alpha) \tag{1}$$

This is the equation for the position of a damped oscillator with respect to time. A, B, γ, ω, and α are parameters that change the shape of the curve plotted by this function in various ways. The function defined on line 36 takes these parameters, as well as t, and returns a value for the position.

In order to begin fitting a curve to this data, we first need a set of initial parameters. These are defined on line 20 and were obtained by guessing a few and then adjusting them until we reached a curve that very roughly fit the data points. They are defined in an array in order to be passed to the function that we'll be using later to properly fit the curve to the data. Below, in Figure 1, you can see the curve of our initial guess along with the values of each parameter in Table 1.

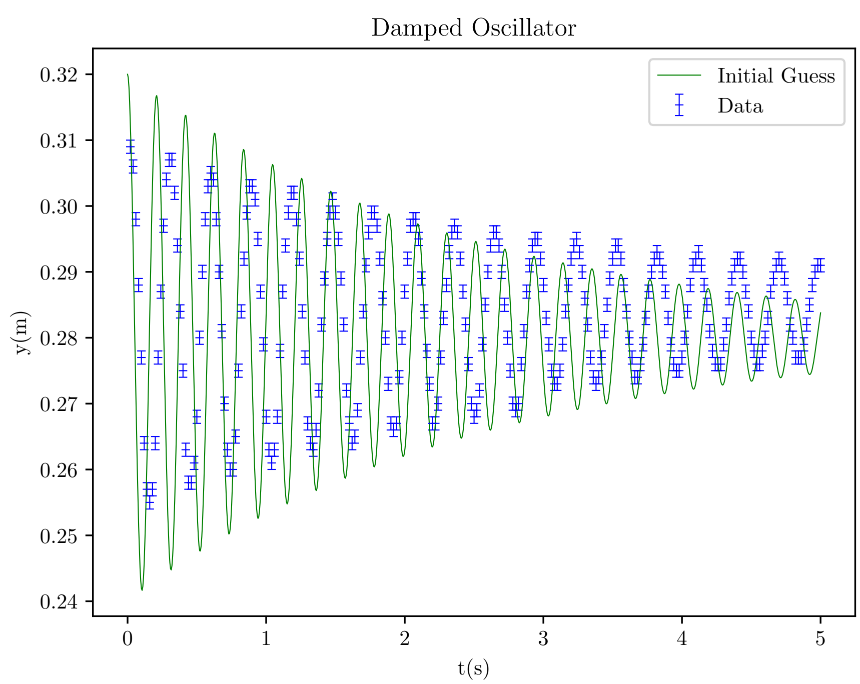

Figure 1: Initial Guess Curve

A	B	γ	ω	α
0.28	0.04	0.4	30	0

Table 1: Initial Guess Parameters

As you can see, the curve fits reasonably well to the data points. It has roughly the same frequency, initial amplitude, and decay rate, which means it's a good initial guess for our algorithm to start with.

The algorithm we used to fit the function to the set of data points was the Levenberg-Marquardt Algorithm, implemented in the `scipy.optimize` module, specifically the function `curve_fit`. The implementation of this function [lines 47-49] is slightly complex but, after providing it with the function we defined earlier, the data points we are considering, and out initial guesses for the parameters, it returns an array of parameters that it determines to be the optimal parameters to use in order to approximately fit the curve to the data. We then feed these parameters back to our original function and it gives us values for $y(t)$ that are very close to correct. The "goodness" of these values is discussed later on. For now we can have a look at the plot of this curve [Figure 2] and see that it seems to be reasonably accurate.

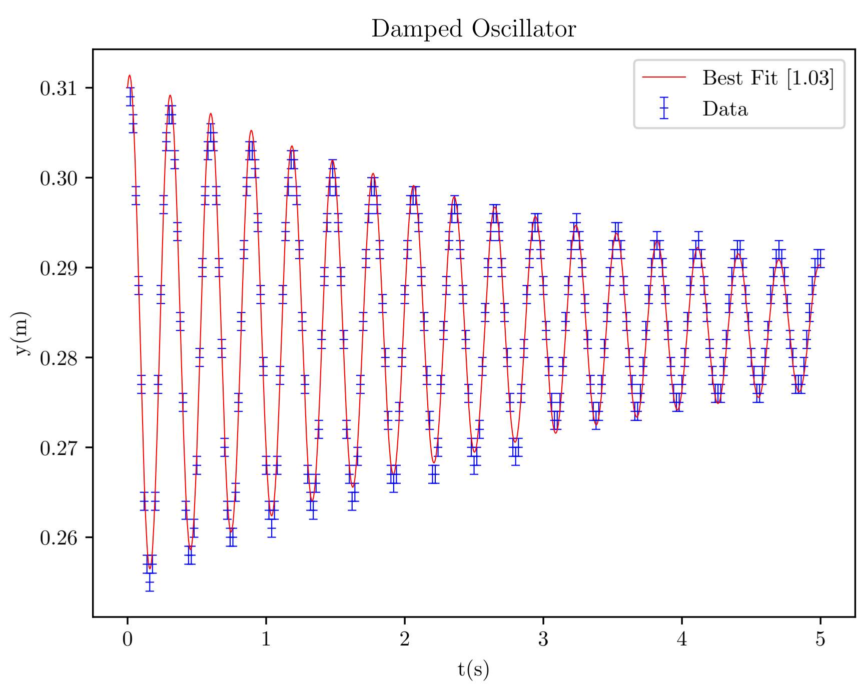

Figure 2: Algorithmically Determined Best Fit

As mentioned before, this line of best fit comes with a so-called "goodness" that we call χ^2 where

$$\chi^2 = \sum_{i=0}^{n} \frac{y_i - f(t_i, A, B, \gamma, \omega, \alpha)}{u_i} \tag{2}$$

This χ^2, calculated on lines 52-53, when divided by `dof`, the "degrees of freedom" of the data (i.e. the number of data points minus the number of fitted parameters [line 54]) gives us a much more useful and convenient measure of the fit. The results of these calculations are below in Table 2.

χ^2	`dof`	$\frac{\chi^2}{\texttt{dof}}$
252.73	245	1.03

Table 2: Goodness Parameters

Our value for $\frac{\chi^2}{\texttt{dof}}$ is ~ 1.03 and ideally this value is ~ 1, so our fit is fairly reasonable. The next thing to consider is the uncertainties of the actual parameters used in the final fit. These can be extracted from the covariance matrix `pcov`, which comes out of the calling of `curve_fit` in line 47. We introduce a correction factor of $\sqrt{\frac{\chi^2}{\texttt{dof}}}$ as $\frac{\chi^2}{\texttt{dof}}$ deviates from 1. These calculations can be found in lines 62-63, resulting in Table 3.

A	B	γ	ω	α
0.28	-0.028	0.28	21.46	-2.81
$\pm 6.4 \times 10^{-5}$	$\pm 2.47 \times 10^{-4}$	$\pm 4.61 \times 10^{-3}$	$\pm 4.66 \times 10^{-3}$	$\pm 8.87 \times 10^{-3}$

Table 3: Parameter Values

The results of line 65, i.e. the uncertainties of the parameters without the correction factor, can be seen below in Table 4.

$u(A)$	$u(B)$	$u(\gamma)$	$u(\omega)$	$u(\alpha)$
6.33×10^{-5}	2.43×10^{-4}	4.54×10^{-3}	4.59×10^{-3}	8.73×10^{-3}

Table 4: Uncertainties of Parameters

Finally, we consider the relationship between each parameter and their correlations as these parameters do not exist isolated from everything else. They are all coupled in some way and in order to see the degree to which they are correlated, we calculate the correlation matrix [lines 68-75], displayed below in Table 5. These values are in the interval [-1, 1] and the matrix is symmetric, so we didn't show all of it.

	A	B	γ	ω	α
A	1				
B	0.00039	1			
γ	0.0027	-0.77	1		
ω	-0.043	0.023	-0.015	1	
α	-0.044	0.032	-0.023	0.77	1

Table 5: Parameter Correlation Matrix

From this table we can see that the strongest correlations are between γ and B, and between ω and α, both of which have a correlation of $\sim$ 0.77. This is far greater than any other relationship in the system as the rest are an order of magnitude less than these two, at least. These high correlation values are significant when calculating uncertainties as two highly correlated values require a more sophisticated method in order to more reliably calculate their uncertainties. For now we can say that this is a fairly good fit to the data and, excusing the two correlations mentioned earlier, the uncertainties in Table 3 are valid.

The second section of this activity was an introduction into a weighted linear least-squares fit. The code for all of this is in Appendix 2 and 3.To begin with, we went back to a section of CP1 and replotted the line of best fit for the data in LinearNoErrors.txt using the `curve_fit` function and got the result below in Figure 3 as well as the tables below showing the parameters and their uncertainties using the same techniques as for the first analysis.

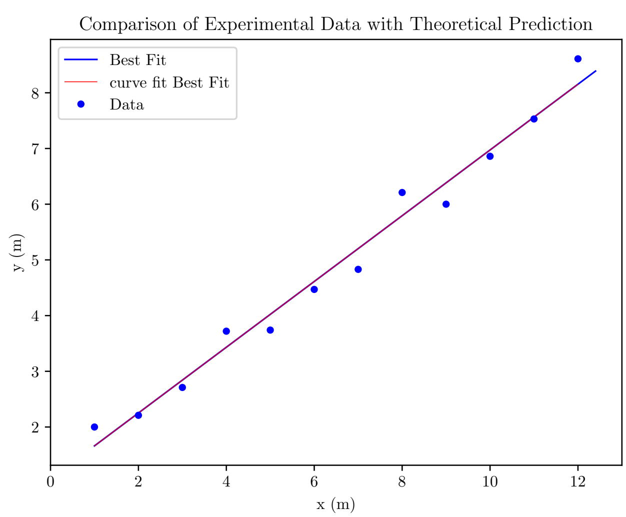

Figure 3: Unweighted Linear Least-Squares Fit for LinearNoErrors.txt

χ^2	`dof`	$\frac{\chi^2}{\texttt{dof}}$
0.99	10	0.099

Table 6: CP1c Goodness Parameters

m	c
0.59	1.07
±0.026	±0.19

Table 7: CP1c Parameter Values

$u(m)$	$u(c)$
0.084	0.62

Table 8: CP1c Unweighted Uncertainties of Parameters

	m	c
m	1	
c	-0.88	1

Table 9: CP1c Parameter Correlation Matrix

m	c
0.59	1.07
±0.026	±0.19

Table 10: CP1c Old Uncertainties

Looking at the values returned using the new form of analysis, we can see that they are exactly the same as the results printed out in lines 49-52 of Appendix 3 [Table 10]. This confirms that `scipy` correctly calculates uncertainties of parameters if the data has no uncertainty. Regarding the analysis of the parameters, the value for $\frac{\chi^2}{\texttt{dof}}$ is quite different from 1, meaning that the fit is not a very good one. This can be seen again in the uncertainties for m and c in the fact that they are each $\sim 10\%$ of the value themselves [Table 7]. Even worse is the fact that the correlation between each value is -0.88 [Table 9], meaning these uncertainties are not as accurate as they should be as a strong correlation between parameters requires more sophisticated methods for calculating uncertainties. In terms of the look of the red line of best fit in Figure 3, it is exactly the same as the blue plot, our previous plot, which further shows that the methods return the same results. The uncertainties don't seem The code for these calculations is on lines 78-102 of Appendix 3.

In order to properly understand what the correlation between parameters means, we use a contour plot, plotting m against c with the corresponding $\frac{\chi^2}{\texttt{dof}}$ for each point. The section of the code that plots the contour is on lines 49-59 of Appendix 2. Below, [Figure 4] is the actual plot for the weighted linear least-squares fit, along with the contour plot corresponding to it [Figure 5].

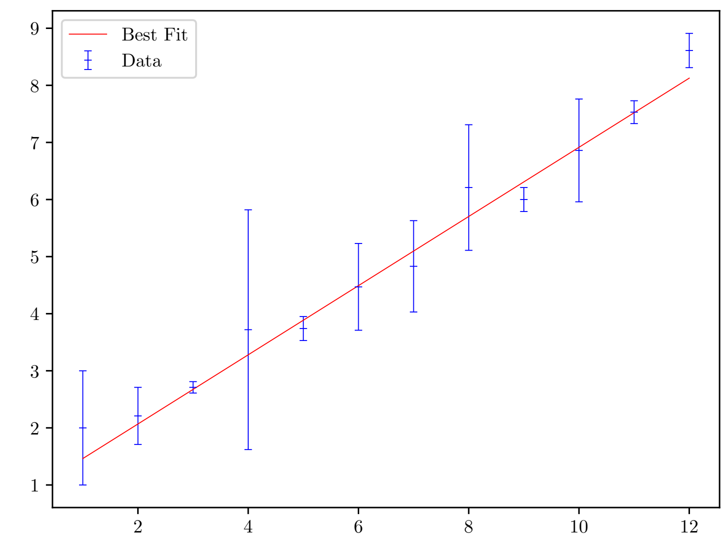

Figure 4: Weighted Linear Least-Squares Fit for LinearWithErrors.txt

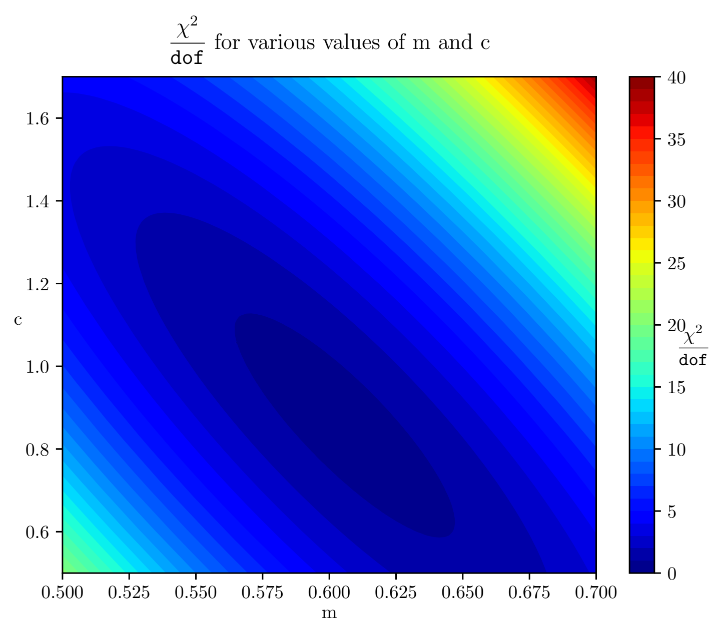

Figure 5: Weighted Contour Plot for LinearWithErrors.txt

Lastly, in Figure 6 we have a contour plot that corresponds to the data from LinearNoErrors.txt. It's clear that the range of possible values of $\frac{\chi^2}{\mathtt{dof}}$ for the weighted fit is much larger than for the unweighted fit. We've plotted the two contours using the same scale for the value of $\frac{\chi^2}{\mathtt{dof}}$ in order to show just how much bigger the range is for the weighted fit.

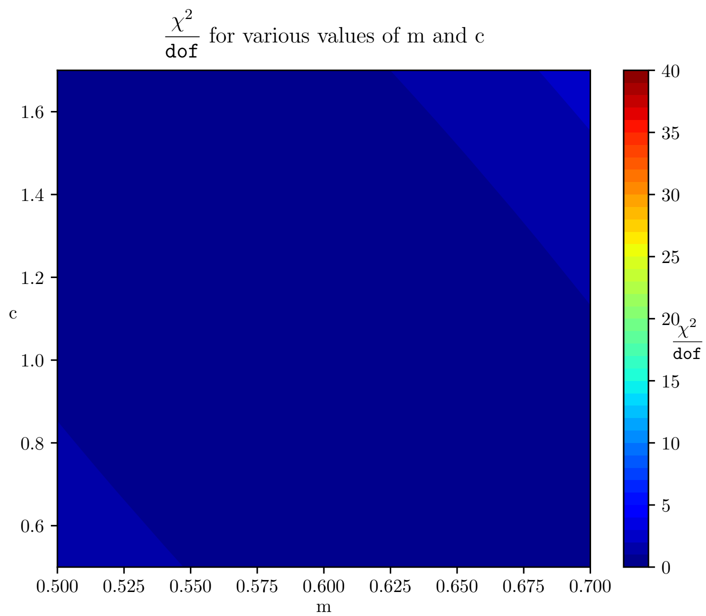

Figure 6: CP1c Unweighted Contour for LinearNoErrors.txt

The values for the unweighted fit are ~ 0 for most values considered and ~ 1 for two small sections in the top right and bottom left. The ideal value for $\frac{\chi^2}{\mathtt{dof}}$ is 1 and so we can see that an unweighted fit gives us a much more accurate fit, but in reality data without errors is impossible and so all fits should be weighted. Thus, when fitting a line to some data, it is vital to properly consider the uncertainties of everything, including all the parameters and the correlations between them, in order to be confident that the results are the actual results, even if the so-called "goodness" factor is less than ideal.

2 Appendix

Appendix 1: CP2a_Nonlinear_Fitting.py

```python
from matplotlib import pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
import matplotlib
matplotlib.use("pgf")
matplotlib.rcParams.update({
    "pgf.texsystem": "pdflatex",
    'font.family': 'serif',
    'text.usetex': True,
    'pgf.rcfonts': False,
})
# File reading and initialisation of variables
file = open('PHY2004W Computational\CP2\DampedData1.txt', 'r')
header = file.readline()
lines = file.readlines()
i = 0
N = len(lines)
data = np.zeros((2, N))
u = [0.001]*N
p0 = [0.28, 0.04, 0.4, 30, 0] # My best guess for the parameters, by
    observation
name = ['A', 'B', 'gamma', 'omega', 'alpha']

for line in lines:
    line = line.strip()
    columns = line.split()
    data[0, i] = float(columns[0])
    data[1, i] = float(columns[1])
    i += 1
file.close()
# Plots the data
plt.errorbar(data[0], data[1], u, fmt='_b', lw=0.5, capsize=2,
    capthick=0.5, markersize=4, markeredgewidth=0.5, label='Data')
plt.title('Damped Oscillator')
plt.xlabel('t(s)')
plt.ylabel('y(m)')
# Defines the function that the curve_fit function uses
def f(t, A, B, gamma, omega, alpha):
    return A+(B*np.exp(-gamma*t))*np.cos((omega*t)-alpha)

# Plots my best guess
tmodel = np.linspace(0.0, 5.0, 1000)
ystart = f(tmodel, *p0)
plt.plot(tmodel, ystart, '-g', lw=0.5, label='Initial Guess')
plt.legend()
plt.savefig('PHY2004W Computational/CP2/CP2a_Initial_Guess.pgf')
```

```python

# Plots the Levenberg-Marquardt best fit
popt, pcov = curve_fit(f, data[0], data[1], p0, sigma=u, absolute_sigma=True)
yfit = f(tmodel, *popt)
plt.plot(tmodel, yfit, '-r', lw=0.5, label='Best Fit [1.03]')

# Calculates chi squared and does magic to work out the fit paramters
dymin = (data[1]-f(data[0], *popt))/u
min_chisq = sum(dymin*dymin)
dof = len(data[0]) - len(popt)

print('Chi Squared:', round(min_chisq, 5))
print('Number of Degrees of Freedom:', round(dof, 5))
print('Chi Squared per Degree of Freedom:', round(min_chisq/dof, 5))
print()

print('Fitted paramters with 68% C.I.:')
for i, pmin in enumerate(popt):
    print('%2i %-10s %12f +/- %10f'%(i, name[i], pmin, np.sqrt(pcov[i,i])*np.sqrt(min_chisq/dof)))
print()
perr = np.sqrt(np.diag(pcov))
print('Perr:', perr)
# Calculates and prints the Correlation matrix
print('Correlation matrix:')
print('          ', end='')
for i in range(len(popt)): print('%10s'%(name[i],), end=''),
print()
for i in range(len(popt)):
    print('%10s'%(name[i]), end=''),
    for j in range(i+1):
        print('%10f'%(pcov[i,j]/np.sqrt(pcov[i,i]*pcov[j,j]),), end=''),
    print()
# Finally saves the best fit curve
plt.legend()
plt.savefig('PHY2004W Computational/CP2/CP2a_Best_Fit.pgf')
```

Appendix 2: CP2b_Visualising_Uncertainties.py

```python
from matplotlib import pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
import matplotlib
matplotlib.use("pgf")
matplotlib.rcParams.update({
    "pgf.texsystem": "pdflatex",
    'font.family': 'serif',
    'text.usetex': True,
```

```
    'pgf.rcfonts': False,
})
# File reading and initialisation of variables
file = open('PHY2004W Computational\CP2\LinearWithErrors.txt', 'r')
header = file.readline()
lines = file.readlines()
i = 0
N = len(lines)
data = np.zeros((3, N))
p0 = np.array([1, 1])
name = np.array(['m', 'c'])
for line in lines:
    line = line.strip()
    columns = line.split()
    data[0, i] = float(columns[0])
    data[1, i] = float(columns[1])
    data[2, i] = float(columns[2])
    i += 1
file.close()
# Defines the function that the curve_fit function uses
def f(x, m, c):
    return m*x+c
# Plots the Levenberg-Marquardt best fit
popt, pcov = curve_fit(f, data[0], data[1], p0, sigma=data[2], absolute_sigma=True)
dof = len(data[1])-len(popt)
# Initialises variables used for plotting the contour plot and plots it
Npts = 10000
mscan = np.zeros(Npts)
cscan = np.zeros(Npts)
chi_dof = np.zeros(Npts)
ncols = 400
c = 0
for mpar in np.linspace(0.5, 0.7, 100, True):
    for cpar in np.linspace(0.5, 1.7, 100, True):
        mscan[c] = mpar
        cscan[c] = cpar
        dymin = (data[1]-f(data[0], mpar, cpar))/data[2]
        chi_dof[c] = sum(dymin*dymin)/dof
        c += 1
plt.figure(1)
# Plots the contour and saves it
plt.title('$\\frac{\\chi^2}{\\texttt{dof}}$ for various values of m and c', pad=15)
plt.xlabel('m')
plt.ylabel('c', rotation = 0)
cntPlt = plt.tricontourf(mscan, cscan, chi_dof, ncols, cmap='jet', levels=np.linspace(0, 40, 41))
for r in cntPlt.collections:
    r.set_edgecolor("face")
```

```
cbar = plt.colorbar()
cbar.set_label('$\\frac{\\chi^2}{\\texttt{dof}}$', rotation=0)
plt.savefig('PHY2004W Computational\CP2\CP2b_Contour_Plot.pgf')

plt.figure(2)
plt.errorbar(data[0], data[1], data[2], fmt='_b', lw=0.5, capsize=2, capthick=0.5, markersize=4, markeredgewidth=0.5, label='Data')
# Beginning of analysis of parameters
tmodel = np.linspace(1, 12, 1000)
yfit = f(tmodel, *popt)
plt.plot(tmodel, yfit, '-r', lw=0.5, label='Best Fit')
# Calculates chi squared and does magic to work out the fit paramters
dymin = (data[1]-f(data[0], *popt))/data[2]
min_chisq = sum(dymin*dymin)
dof = len(data[0]) - len(popt)

print('Chi Squared:', round(min_chisq, 5))
print('Number of Degrees of Freedom:', round(dof, 5))
print('Chi Squared per Degree of Freedom:', round(min_chisq/dof, 5))
print()

print('Fitted paramters with 68% C.I.:')
for i, pmin in enumerate(popt):
    print('%2i %-10s %12f +/- %10f'%(i, name[i], pmin, np.sqrt(pcov[i,i])*np.sqrt(min_chisq/dof)))
print()
perr = np.sqrt(np.diag(pcov))
print('Perr:', perr)
# Calculates and prints the Correlation matrix
print('Correlation matrix:')
print('          ', end='')
for i in range(len(popt)): print('%10s'%(name[i],), end=''),
print()
for i in range(len(popt)):
    print('%10s'%(name[i]), end=''),
    for j in range(i+1):
        print('%10f'%(pcov[i,j]/np.sqrt(pcov[i,i]*pcov[j,j]),), end=''),
    print()

plt.legend()
plt.savefig('PHY2004W Computational\CP2\CP2b_Data_Plot.pgf')
```

Appendix 3: CP1c.py

```
import scipy.stats as stats
from math import sqrt
from matplotlib import pyplot as plt
import matplotlib
from scipy.optimize import curve_fit
```

```
import numpy as np
matplotlib.use("pgf")
matplotlib.rcParams.update({
    "pgf.texsystem": "pdflatex",
    'font.family': 'serif',
    'text.usetex': True,
    'pgf.rcfonts': False,
})

f = open('PHY2004W Computational\CP2\LinearNoErrors.txt', 'r')
header = f.readline()
N = 12
data = np.zeros([3, N])
i = 0
p0 = np.array([1, 1])
name = np.array(['m', 'c'])
for line in f:
    data[0, i] = line.split()[0]
    data[1, i] = line.split()[1]
    data[2, i] = 1
    i += 1
f.close()

xy = []
for c in range(N):
    xy.append(round(data[0, c]*data[1,c], 3))
x2 = []
for t in range(N):
    x2.append(round(data[0,t]**2, 3))

x = data[0]
y = data[1]
d = []
d2 = []
m = ((N*sum(xy)) - sum(x)*sum(y))/((N*sum(x2))-(sum(x))**2)
c = ((sum(x2)*sum(y))-(sum(xy)*sum(x)))/((N*sum(x2))-(sum(x)**2))

for r in range(N):
    d.append(y[r] - ((m*x[r]) + c))
    d2.append(d[r]**2)
um = sqrt(((sum(d2)/((N*sum(x2))-(sum(x)**2)))*(N/(N-2))))
uc = sqrt((((sum(d2)*sum(x2))/(N*((N*sum(x2))-(sum(x)**2))))*(N/(N-2)
    )))

print("m:", round(m, 5))
print("u(m):", round(um, 5))
print("c:", round(c, 5))
print("u(c):", round(uc, 5))
print()
# Plots the data and the line of best fit calculated above
xLine = np.arange(1, 12.5, 0.1)
```

```
yLine = []
for i in xLine:
    yLine.append((m*i)+c)
plt.figure(1)
plt.plot(xLine, yLine, color='blue', label="Best Fit", lw=1)
plt.errorbar(data[0], data[1], fmt='ob', lw=0.5, capsize=2, capthick=0.5, markersize=4, markeredgewidth=0.5, label='Data')
# Plots the line of best fit using curve_fit
def f(x, m, c):
    return (m*x)+c
popt, pcov = curve_fit(f, data[0], data[1], p0, sigma=data[2], absolute_sigma=True)
dof = len(y)-len(popt)
tmodel = np.linspace(1, 12, 1000)
ystart = f(tmodel, *p0)
yfit = f(tmodel, *popt)
plt.plot(tmodel, yfit, '-r', lw=0.5, label='curve fit Best Fit')
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.title("Comparison of Experimental Data with Theoretical Prediction")
plt.xlim(0,13)
plt.legend()
plt.savefig('PHY2004W Computational\CP2\CP1c_Data_Plot.pgf')
# Calculates chi squared and does magic to work out the fit paramters
dymin = (y-f(x, *popt))/data[2]
min_chisq = sum(dymin*dymin)
dof = len(x) - len(popt)

print('Chi Squared:', round(min_chisq, 5))
print('Number of Degrees of Freedom:', round(dof, 5))
print('Chi Squared per Degree of Freedom:', round(min_chisq/dof, 5))
print()

print('Fitted paramters with 68% C.I.:')
for i, pmin in enumerate(popt):
    print('%2i %-10s %12f +/- %10f'%(i, name[i], pmin, np.sqrt(pcov[i,i])*np.sqrt(min_chisq/dof)))
print()
perr = np.sqrt(np.diag(pcov))
print('Perr:', perr)
# Calculates and prints the Correlation matrix
print('Correlation matrix:')
print('          ', end='')
for i in range(len(popt)): print('%10s'%(name[i],), end=''),
print()
for i in range(len(popt)):
    print('%10s'%(name[i]), end=''),
    for j in range(i+1):
        print('%10f'%(pcov[i,j]/np.sqrt(pcov[i,i]*pcov[j,j]),), end=''),
```

```
    print()
# Initialises variables
Npts = 10000
mscan = np.zeros(Npts)
cscan = np.zeros(Npts)
chi_dof = np.zeros(Npts)
ncols = 25
c = 0
for mpar in np.linspace(0.5, 0.7, 100, True):
    for cpar in np.linspace(0.5, 1.7, 100, True):
        mscan[c] = mpar
        cscan[c] = cpar
        dymin = (data[1]-f(data[0], mpar, cpar))/data[2]
        chi_dof[c] = sum(dymin*dymin)/dof
        c += 1
plt.figure(2)
# Plots the contour and saves it
plt.title('$\\frac{\\chi^2}{\\texttt{dof}}$ for various values of m and c', pad=15)
plt.xlabel('m')
plt.ylabel('c', rotation = 0)
cntPlt = plt.tricontourf(mscan, cscan, chi_dof, ncols, cmap='jet', levels=np.linspace(0, 40, 41))
for r in cntPlt.collections:
    r.set_edgecolor("face")
cbar = plt.colorbar()
cbar.set_label('$\\frac{\\chi^2}{\\texttt{dof}}$', rotation=0)
plt.savefig('PHY2004W Computational\CP2\CP1c_Contour_Plot.pgf')
```