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 33 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.

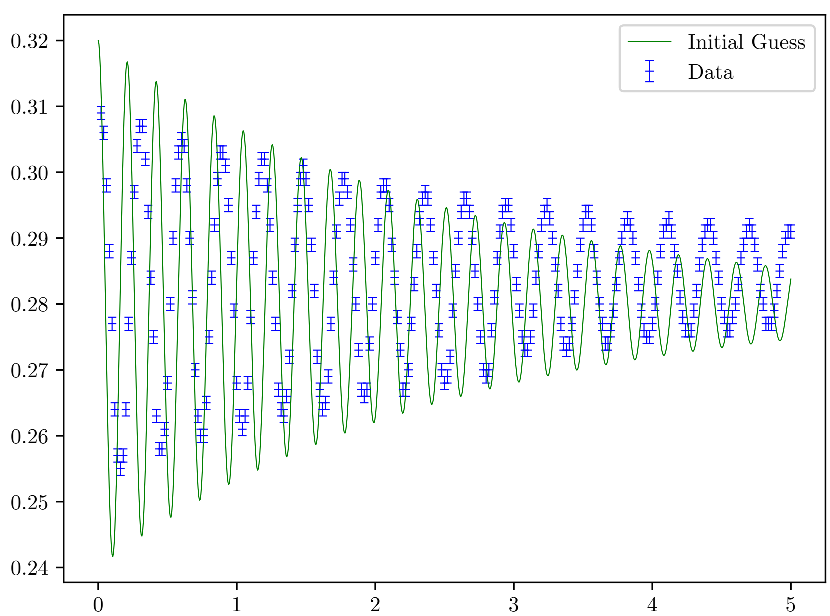

Figure 1: Initial Guess Curve

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 44-46] 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 "correctness" of these values are 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 acceptably accurate.

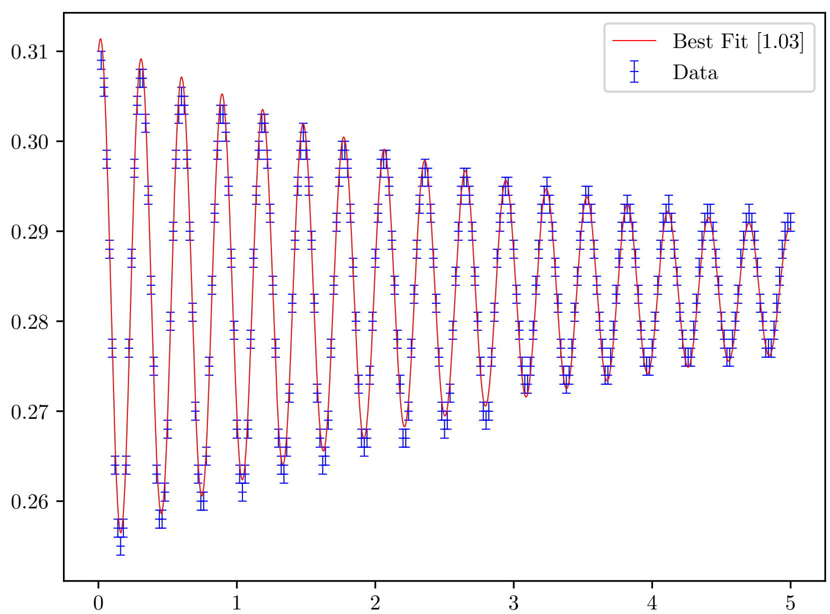

Figure 2: Algorithmically Determined Best Fit

2 Appendix

```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')
# 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')

# 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 1: CP2a_Nonlinear_Fitting.py

```
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()
```

```python
i = 0
N = len(lines)
data = np.zeros((3, N))
p0 = [1, 1] # Initial guess, not that significant as long as it's reasonable
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 = 1000
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()
# Plots the contour and saves it
plt.tricontourf(mscan, cscan, chi_dof, ncols)
plt.colorbar()
plt.savefig('PHY2004W Computational\CP2\CP2b_Contour_Plot.pgf')
```

Appendix 2: CP2b_Visualising_Uncertainties.py