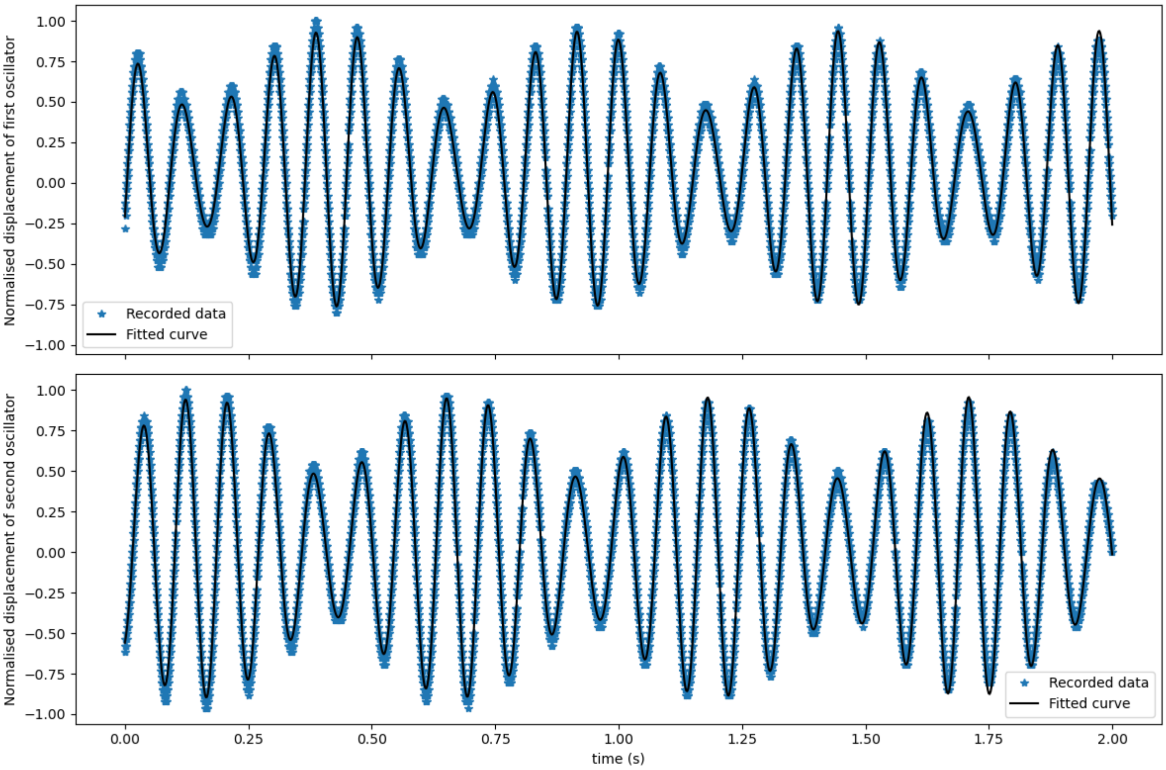
In [ ]:
```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
from uncertainties import ufloat, unumpy
from scipy.signal import find_peaks

def oscillator(t, A, B, omega_plus, omega_minus, phi, d, C):
    return A*np.cos(omega_plus*(t-d)+phi) + B*np.cos(omega_minus*(t-d)+ph

filenames = []
for i in range(1, 51):
    filename = "/home/ws10/maitrey/p442-integrated-lab/IV. Coupled Oscill
    filenames.append(filename)


for i, file1 in enumerate(filenames):
    data = np.genfromtxt(file1, skip_header=18, delimiter=',')
    data[:, 0] = data[:, 0] - data[:, 0][0]
    data[:, 1] = data[:, 1] / np.max(data[:, 1])
    data[:, 2] = data[:, 2] - data[:, 2][0]
    data[:, 3] = data[:, 3] / np.max(data[:, 3])
    np.savetxt('DLL{:04d}'.format(i+1) + '.CSV', data, delimiter=',')


# filename = '/home/ws10/maitrey/p442-integrated-lab/IV. Coupled Oscillat
# data = np.genfromtxt(filename, skip_header=18, delimiter=',')


# # t1 = data[:, 0] # first wave
# data[:, 0] = data[:, 0] - data[:, 0][0]
# # y1 = data[:, 1] # first wave
# data[:, 1] = data[:, 1] / np.max(data[:, 1])
# # t2 = data[:, 2] # second wave
# data[:, 2] = data[:, 2] - data[:, 2][0]
# # y2 = data[:, 3] # second wave
# data[:, 3] = data[:, 3] / np.max(data[:, 3])

# Create an empty list to store fit parameters for each file
fit_params1 = []
fit_params2 = []
fit_params1_u = []
```

In [ ]:
```
file1 = '/home/ws10/maitrey/p442-integrated-lab/IV. Coupled Oscillator/DK
data = np.loadtxt(file1, delimiter=',')
t1 = data[:, 0] # first wave
y1 = data[:, 1] # first wave
t2 = data[:, 2] # second wave
y2 = data[:, 3] # second wave
sigma = np.ones(len(t1))*0.0004
guess = (0.617704, 0.243294, 70.7494, 84.239, 319.04, -0.0768066, 0.08589
# print(t1, y1)

popt1, pcov1 = curve_fit(oscillator, t1, y1, sigma=sigma, p0=guess)
popt2, pcov2 = curve_fit(oscillator, t2, y2, sigma=sigma, p0=guess)
print(popt1)
perr1 = np.sqrt(np.diag(pcov1))
perr2 = np.sqrt(np.diag(pcov2))
popt1_u = [ufloat(p, e) for p, e in zip(popt1, perr1)]
```

```
popt2_u = [ufloat(p, e) for p, e in zip(popt2, perr2)]
print(popt1_u)
A_1, B_1, omega_plus_1, omega_minus_1, phi_1, d_1, C_1 = popt1
A_2, B_2, omega_plus_2, omega_minus_2, phi_2, d_2, C_2 = popt2
A_1_u, B_1_u, omega_plus_1_u, omega_minus_1_u, phi_1_u, d_1_u, C_1_u = pc
A_2_u, B_2_u, omega_plus_2_u, omega_minus_2_u, phi_2_u, d_2_u, C_2_u = pc
fit_params1.append(popt1_u)
fit_params2.append(popt2_u)
# A_1 = abs(A_1)
# B_1 = abs(B_1)
# A_2 = abs(A_2)
# B_2 = abs(B_2)
DELTA_1_u = (omega_plus_1_u - omega_minus_1_u) * (abs(B_1_u) - abs(A_1_u)
OMEGA_1_u = (omega_plus_1_u - omega_minus_1_u) * (1 - ((abs(B_1_u) - abs(
omega_g_u = (omega_plus_1_u + omega_minus_1_u - DELTA_1_u) / 2 # Ground S
DELTA_2_u = (omega_plus_2_u - omega_minus_2_u) * (abs(B_2_u) - abs(A_2_u)
OMEGA_2_u = (omega_plus_2_u - omega_minus_2_u) * (1 - ((abs(B_2_u) - abs(
omega_e_u = (omega_plus_2_u + omega_minus_2_u - DELTA_2_u) / 2 # Excited
DELTA_u = (DELTA_1_u + DELTA_2_u) / 2 # Average Detuning
OMEGA_u = (OMEGA_1_u + OMEGA_2_u) / 2 # Average Rabi Frequency
GEOMETRIC_PHASE_u = np.pi * (1 - (DELTA_u / (unumpy.sqrt(OMEGA_u**2 + DEL
fit_params1_u.append([DELTA_u, GEOMETRIC_PHASE_u/np.pi])
print(DELTA_u, GEOMETRIC_PHASE_u/np.pi)
curve_t1 = np.linspace(0, 2, 1000)
curve_y1 = oscillator(curve_t1, A_1, B_1, omega_plus_1, omega_minus_1, ph
curve_t2 = np.linspace(0, 2, 1000)
curve_y2 = oscillator(curve_t2, A_2, B_2, omega_plus_2, omega_minus_2, ph
# print(curve_y2)
fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(12, 8), sharex=True, sha

ax1.plot(t1, y1, '*', label='Recorded data')
ax1.plot(curve_t1, curve_y1, 'black', label='Fitted curve')
# ax1.set_xlabel('time (s)')
ax1.set_ylabel('Normalised displacement of first oscillator')
ax1.legend()

ax2.plot(t2, y2, '*', label='Recorded data')
ax2.plot(curve_t2, curve_y2, 'black', label='Fitted curve')
ax2.set_xlabel('time (s)')
ax2.set_ylabel('Normalised displacement of second oscillator')
ax2.legend()


fig.align_ylabels()
plt.tight_layout()
# fig.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05), ncol=2)
plt.savefig(f"{file1[-6:-4]}.png",dpi=600)
plt.show()
print(pcov1)
```

```
[ 6.02832851e-01  2.49472640e-01  7.12286778e+01  8.33260685e+01
  3.16989030e+02 -1.03993326e-01  8.62407749e-02]
[0.6028328505761817+/-0.0008766872438814897, 0.24947263953669144+/-0.00087
56776010593886, 71.22867776252346+/-0.0025066616016685903, 83.326068456418
14+/-0.006094205937276608, 316.989029532129+/-0.04746256212381065, -0.1039
9332560862665+/-0.0006526268653386184, 0.08624077494866184+/-0.00061553708
00141696]
5.283+/-0.013 0.5637+/-0.0010
```

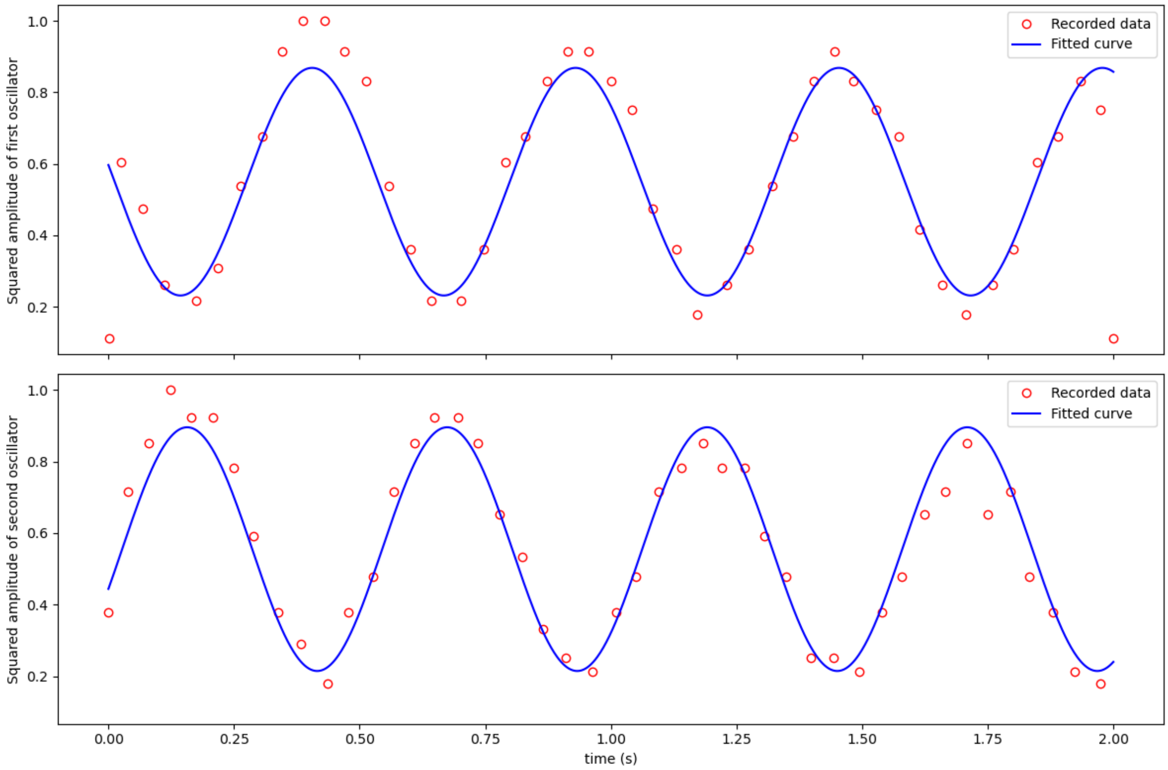



```
[[ 7.68580524e-07  2.36824478e-08  1.51761995e-08 -4.56947517e-07
  -2.48776918e-06 -3.48929193e-08 -9.42783208e-09]
 [ 2.36824478e-08  7.66811261e-07  2.24914950e-07  2.02813486e-08
  -1.15823362e-06 -1.32197830e-08 -7.19123808e-09]
 [ 1.51761995e-08  2.24914950e-07  6.28335239e-06  6.42283112e-07
  -3.85895795e-05 -4.44334319e-07  2.37039720e-08]
 [-4.56947517e-07  2.02813486e-08  6.42283112e-07  3.71393460e-05
   2.32215731e-04  3.28217440e-06  1.70175054e-08]
 [-2.48776918e-06 -1.15823362e-06 -3.85895795e-05  2.32215731e-04
   2.25269480e-03  3.09135290e-05 -1.95585048e-07]
 [-3.48929193e-08 -1.32197830e-08 -4.44334319e-07  3.28217440e-06
   3.09135290e-05  4.25921825e-07 -2.32283158e-09]
 [-9.42783208e-09 -7.19123808e-09  2.37039720e-08  1.70175054e-08
  -1.95585048e-07 -2.32283158e-09  3.78885897e-07]]
```

In [ ]:

```
file1 = '/home/ws10/maitrey/p442-integrated-lab/IV. Coupled Oscillator/DK
data = np.loadtxt(file1, delimiter=',')
t1 = data[:, 0] # first wave
y1 = data[:, 1]-0.1 # first wave
t2 = data[:, 2] # second wave
y2 = data[:, 3] # second wave

y1_square = y1**2
# Normalise the data
y1_square = y1_square / max(y1_square)
peaks1, _ = find_peaks(y1_square, height=0.1, distance=60)
y2_square = y2**2
# Normalise the data
y2_square = y2_square / max(y2_square)
peaks2, _ = find_peaks(y2_square, height=0.1, distance=60)


def sine_squared(x, A, omega, phi, C):
    return A * np.sin(omega * x + phi)**2 + C

t1_peaks = t1[peaks1]
y1_peaks = y1_square[peaks1]
```

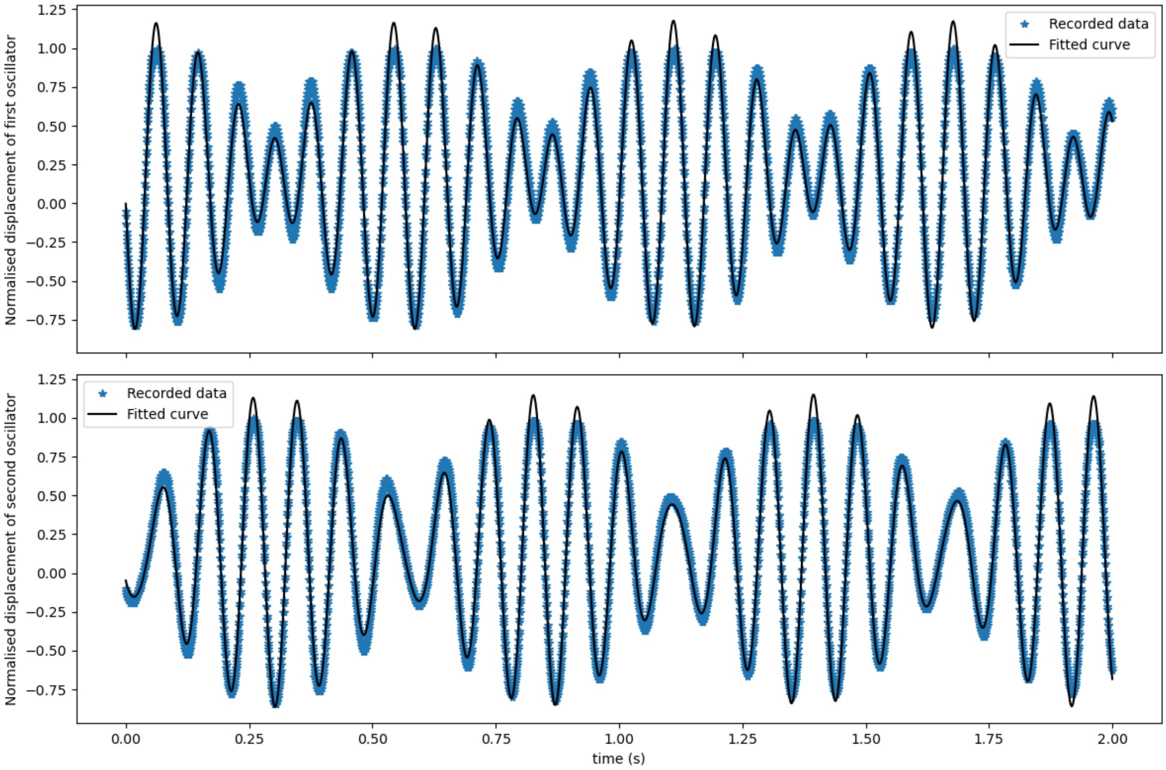
```
t2_peaks = t2[peaks2]
y2_peaks = y2_square[peaks2]

# Write squared amplitude data to file as (x,y) pairs
np.savetxt('test_please.txt', np.transpose([t1_peaks, y1_peaks]), delimit

popt1, _ = curve_fit(sine_squared, t1_peaks, y1_peaks, p0=[-0.636255, 5.9
popt2, _ = curve_fit(sine_squared, t2_peaks, y2_peaks, p0=[-0.636255, 5.9

# Plot the fitted curves

# first intialise point for fitted function
curve_t1 = np.linspace(0, 2, 1000)
curve_y1 = sine_squared(curve_t1, *popt1)
curve_t2 = np.linspace(0, 2, 1000)
curve_y2 = sine_squared(curve_t2, *popt2)
fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(12, 8), sharex=True, sha

ax1.plot(t1_peaks, y1_peaks, 'o', label='Recorded data', markerfacecolor=
ax1.plot(curve_t1, curve_y1, 'blue', label='Fitted curve')
# ax1.set_xlabel('time (s)')
ax1.set_ylabel('Squared amplitude of first oscillator')
ax1.legend()

ax2.plot(t2_peaks, y2_peaks, 'o', label='Recorded data', markerfacecolor=
ax2.plot(curve_t2, curve_y2, 'blue', label='Fitted curve')
ax2.set_xlabel('time (s)')
ax2.set_ylabel('Squared amplitude of second oscillator')
ax2.legend()


fig.align_ylabels()
plt.tight_layout()
# fig.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05), ncol=2)
plt.savefig(f"{file1[-6:-4]}_squared.png",dpi=600)
plt.show()
```



In [ ]:
```
file1 = '/home/ws10/maitrey/p442-integrated-lab/IV. Coupled Oscillator/DK
data = np.loadtxt(file1, delimiter=',')
t1 = data[:, 0] # first wave
y1 = data[:, 1] # first wave
t2 = data[:, 2] # second wave
y2 = data[:, 3] # second wave
sigma = np.ones(len(t1))*0.0004
guess = (0.417384, -0.282505, 66.8972, 77.3538, -3.59342, -0.0555561,0.38
# print(t1, y1)

popt1, pcov1 = curve_fit(oscillator, t1, y1, sigma=sigma, p0=guess)
popt2, pcov2 = curve_fit(oscillator, t2, y2, sigma=sigma, p0=guess)
print(popt1)
perr1 = np.sqrt(np.diag(pcov1))
perr2 = np.sqrt(np.diag(pcov2))
popt1_u = [ufloat(p, e) for p, e in zip(popt1, perr1)]
popt2_u = [ufloat(p, e) for p, e in zip(popt2, perr2)]
print(popt1_u)
A_1, B_1, omega_plus_1, omega_minus_1, phi_1, d_1, C_1 = popt1
A_2, B_2, omega_plus_2, omega_minus_2, phi_2, d_2, C_2 = popt2
A_1_u, B_1_u, omega_plus_1_u, omega_minus_1_u, phi_1_u, d_1_u, C_1_u = pc
A_2_u, B_2_u, omega_plus_2_u, omega_minus_2_u, phi_2_u, d_2_u, C_2_u = pc
fit_params1.append(popt1_u)
fit_params2.append(popt2_u)
# A_1 = abs(A_1)
# B_1 = abs(B_1)
# A_2 = abs(A_2)
# B_2 = abs(B_2)
DELTA_1_u = (omega_plus_1_u - omega_minus_1_u) * (abs(B_1_u) - abs(A_1_u)
OMEGA_1_u = (omega_plus_1_u - omega_minus_1_u) * (1 - ((abs(B_1_u) - abs(
omega_g_u = (omega_plus_1_u + omega_minus_1_u - DELTA_1_u) / 2 # Ground S
DELTA_2_u = (omega_plus_2_u - omega_minus_2_u) * (abs(B_2_u) - abs(A_2_u)
OMEGA_2_u = (omega_plus_2_u - omega_minus_2_u) * (1 - ((abs(B_2_u) - abs(
omega_e_u = (omega_plus_2_u + omega_minus_2_u - DELTA_2_u) / 2 # Excited
DELTA_u = (DELTA_1_u + DELTA_2_u) / 2 # Average Detuning
OMEGA_u = (OMEGA_1_u + OMEGA_2_u) / 2 # Average Rabi Frequency
```

```
GEOMETRIC_PHASE_u = np.pi * (1 - (DELTA_u / (unumpy.sqrt(OMEGA_u**2 + DEL
fit_params1_u.append([DELTA_u, GEOMETRIC_PHASE_u/np.pi])
print(DELTA_u, GEOMETRIC_PHASE_u/np.pi)
curve_t1 = np.linspace(0, 2, 1000)
curve_y1 = oscillator(curve_t1, A_1, B_1, omega_plus_1, omega_minus_1, ph
curve_t2 = np.linspace(0, 2, 1000)
curve_y2 = oscillator(curve_t2, A_2, B_2, omega_plus_2, omega_minus_2, ph
# print(curve_y2)
fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(12, 8), sharex=True, sha

ax1.plot(t1, y1, '*', label='Recorded data')
ax1.plot(curve_t1, curve_y1, 'black', label='Fitted curve')
# ax1.set_xlabel('time (s)')
ax1.set_ylabel('Normalised displacement of first oscillator')
ax1.legend()

ax2.plot(t2, y2, '*', label='Recorded data')
ax2.plot(curve_t2, curve_y2, 'black', label='Fitted curve')
ax2.set_xlabel('time (s)')
ax2.set_ylabel('Normalised displacement of second oscillator')
ax2.legend()


fig.align_ylabels()
plt.tight_layout()
# fig.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05), ncol=2)
plt.savefig(f"{file1[-6:-4]}.png",dpi=600)
plt.show()
print(pcov1)
```

```
[ 0.37875133 -0.61762316 66.17233722 77.72902702 21.91282826  0.30134846
  0.18180951]
[0.37875132701162645+/-0.0012991163655134235, -0.617623160037611+/-0.00129
88867333221354, 66.17233721754062+/-0.005947813565752505, 77.729027017181
82+/-0.0036532393005249265, 21.91282825796102+/-0.04184757735318713, 0.301
3484564498189+/-0.0005606414339470528, 0.18180951121663336+/-0.00091596832
61164964]
0.302+/-0.015 0.9728+/-0.0013
```



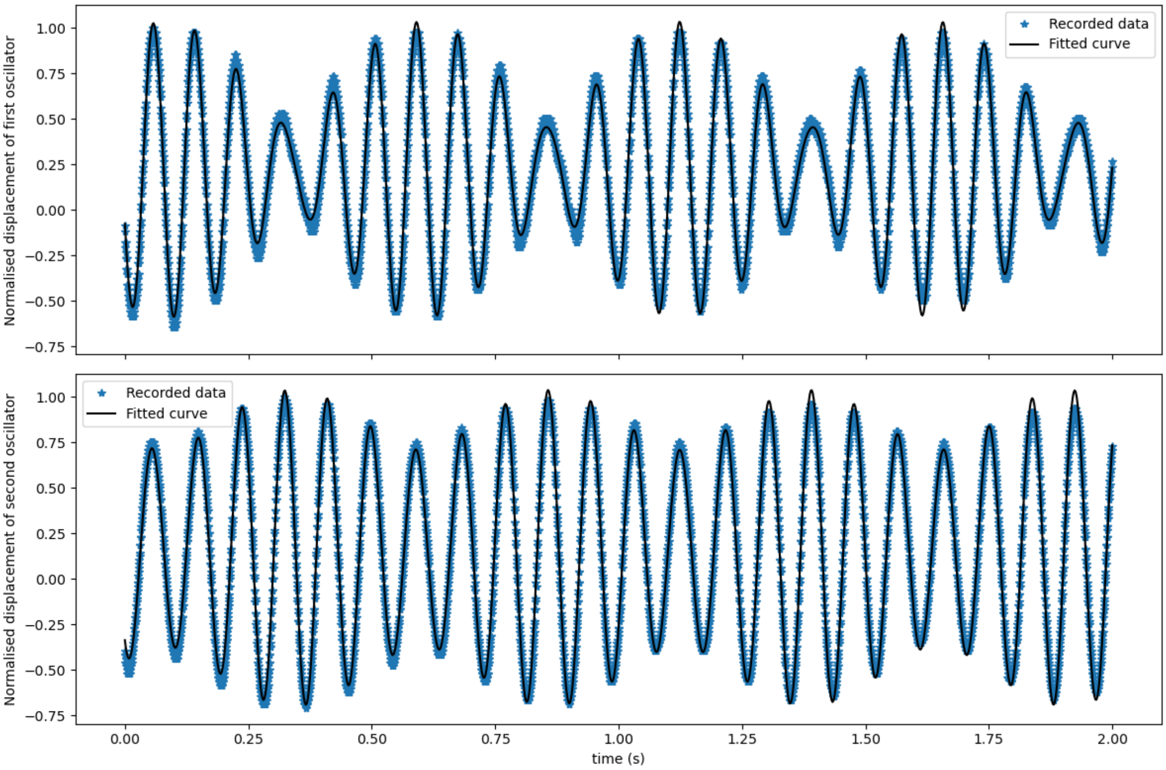

```
[[ 1.68770333e-06 -2.02661883e-08  8.49548385e-08 -4.56011015e-08
  -1.79674496e-06 -2.60450136e-08  2.53380359e-09]
 [-2.02661883e-08  1.68710675e-06 -2.28499917e-07  2.51176053e-08
   3.29321212e-06  4.24378974e-08 -1.14914307e-08]
 [ 8.49548385e-08 -2.28499917e-07  3.53764862e-05  1.13164730e-06
  -1.69080208e-04 -2.17968771e-06  1.01991686e-07]
 [-4.56011015e-08  2.51176053e-08  1.13164730e-06  1.33461574e-05
   5.54351785e-05  8.32567261e-07 -4.71892913e-08]
 [-1.79674496e-06  3.29321212e-06 -1.69080208e-04  5.54351785e-05
   1.75121973e-03  2.34105743e-05 -5.58684713e-07]
 [-2.60450136e-08  4.24378974e-08 -2.17968771e-06  8.32567261e-07
   2.34105743e-05  3.14318817e-07 -7.42163966e-09]
 [ 2.53380359e-09 -1.14914307e-08  1.01991686e-07 -4.71892913e-08
  -5.58684713e-07 -7.42163966e-09  8.38997974e-07]]
```

In [ ]:

```
file1 = '/home/ws10/maitrey/p442-integrated-lab/IV. Coupled Oscillator/DK
data = np.loadtxt(file1, delimiter=',')
t1 = data[:, 0] # first wave
y1 = data[:, 1] # first wave
t2 = data[:, 2] # second wave
y2 = data[:, 3] # second wave
sigma = np.ones(len(t1))*0.0004
guess = (-0.303663, -0.539404, 82.7955,70.4074, -7.07838, 0.0886124,0.212
# print(t1, y1)

popt1, pcov1 = curve_fit(oscillator, t1, y1, sigma=sigma, p0=guess)
popt2, pcov2 = curve_fit(oscillator, t2, y2, sigma=sigma, p0=guess)
print(popt1)
perr1 = np.sqrt(np.diag(pcov1))
perr2 = np.sqrt(np.diag(pcov2))
popt1_u = [ufloat(p, e) for p, e in zip(popt1, perr1)]
popt2_u = [ufloat(p, e) for p, e in zip(popt2, perr2)]
print(popt1_u)
A_1, B_1, omega_plus_1, omega_minus_1, phi_1, d_1, C_1 = popt1
A_2, B_2, omega_plus_2, omega_minus_2, phi_2, d_2, C_2 = popt2
A_1_u, B_1_u, omega_plus_1_u, omega_minus_1_u, phi_1_u, d_1_u, C_1_u = pc
A_2_u, B_2_u, omega_plus_2_u, omega_minus_2_u, phi_2_u, d_2_u, C_2_u = pc
```

```
fit_params1.append(popt1_u)
fit_params2.append(popt2_u)
# A_1 = abs(A_1)
# B_1 = abs(B_1)
# A_2 = abs(A_2)
# B_2 = abs(B_2)
DELTA_1_u = (omega_plus_1_u - omega_minus_1_u) * (abs(B_1_u) - abs(A_1_u)
OMEGA_1_u = (omega_plus_1_u - omega_minus_1_u) * (1 - ((abs(B_1_u) - abs(
omega_g_u = (omega_plus_1_u + omega_minus_1_u - DELTA_1_u) / 2 # Ground S
DELTA_2_u = (omega_plus_2_u - omega_minus_2_u) * (abs(B_2_u) - abs(A_2_u)
OMEGA_2_u = (omega_plus_2_u - omega_minus_2_u) * (1 - ((abs(B_2_u) - abs(
omega_e_u = (omega_plus_2_u + omega_minus_2_u - DELTA_2_u) / 2 # Excited
DELTA_u = (DELTA_1_u + DELTA_2_u) / 2 # Average Detuning
OMEGA_u = (OMEGA_1_u + OMEGA_2_u) / 2 # Average Rabi Frequency
GEOMETRIC_PHASE_u = np.pi * (1 - (DELTA_u / (unumpy.sqrt(OMEGA_u**2 + DEL
fit_params1_u.append([DELTA_u, GEOMETRIC_PHASE_u/np.pi])
print(DELTA_u, GEOMETRIC_PHASE_u/np.pi)
curve_t1 = np.linspace(0, 2, 1000)
curve_y1 = oscillator(curve_t1, A_1, B_1, omega_plus_1, omega_minus_1, ph
curve_t2 = np.linspace(0, 2, 1000)
curve_y2 = oscillator(curve_t2, A_2, B_2, omega_plus_2, omega_minus_2, ph
# print(curve_y2)
fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(12, 8), sharex=True, sha

ax1.plot(t1, y1, '*', label='Recorded data')
ax1.plot(curve_t1, curve_y1, 'black', label='Fitted curve')
# ax1.set_xlabel('time (s)')
ax1.set_ylabel('Normalised displacement of first oscillator')
ax1.legend()

ax2.plot(t2, y2, '*', label='Recorded data')
ax2.plot(curve_t2, curve_y2, 'black', label='Fitted curve')
ax2.set_xlabel('time (s)')
ax2.set_ylabel('Normalised displacement of second oscillator')
ax2.legend()


fig.align_ylabels()
plt.tight_layout()
# fig.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05), ncol=2)
plt.savefig(f"{file1[-6:-4]}.png",dpi=600)
plt.show()
print(pcov1)
```

```
[-2.91750980e-01 -5.21730565e-01  8.26634137e+01  7.06081713e+01
 -7.61266977e+00  8.15893773e-02  2.19780689e-01]
[-0.2917509799376274+/-0.0008210086498802296, -0.5217305652573686+/-0.0008
20608743810809, 82.66341366265827+/-0.004908539638342449, 70.6081712873375
4+/-0.002748163261870378, -7.612669768885203+/-0.0390098032617205, 0.08158
937734435592+/-0.0005277000603470319, 0.21978068868964304+/-0.000579984198
0669023]
5.441+/-0.013 0.5396+/-0.0011
```




```
[[ 6.74055203e-07 -6.51443606e-10  2.18512122e-08 -4.13056939e-08
   7.05551596e-07  8.68350394e-09 -4.81271091e-09]
 [-6.51443606e-10  6.73398710e-07  7.83389351e-10 -1.82442896e-08
   6.65051574e-07  9.20126286e-09 -7.64166916e-09]
 [ 2.18512122e-08  7.83389351e-10  2.40937614e-05  2.32938612e-07
   1.34556088e-04  1.89414505e-06  2.68922086e-08]
 [-4.13056939e-08 -1.82442896e-08  2.32938612e-07  7.55240131e-06
  -5.26247119e-05 -6.46511801e-07  5.97364503e-09]
 [ 7.05551596e-07  6.65051574e-07  1.34556088e-04 -5.26247119e-05
   1.52176475e-03  2.05436953e-05 -7.48546764e-08]
 [ 8.68350394e-09  9.20126286e-09  1.89414505e-06 -6.46511801e-07
   2.05436953e-05  2.78467354e-07 -8.33476280e-10]
 [-4.81271091e-09 -7.64166916e-09  2.68922086e-08  5.97364503e-09
  -7.48546764e-08 -8.33476280e-10  3.36381670e-07]]
```

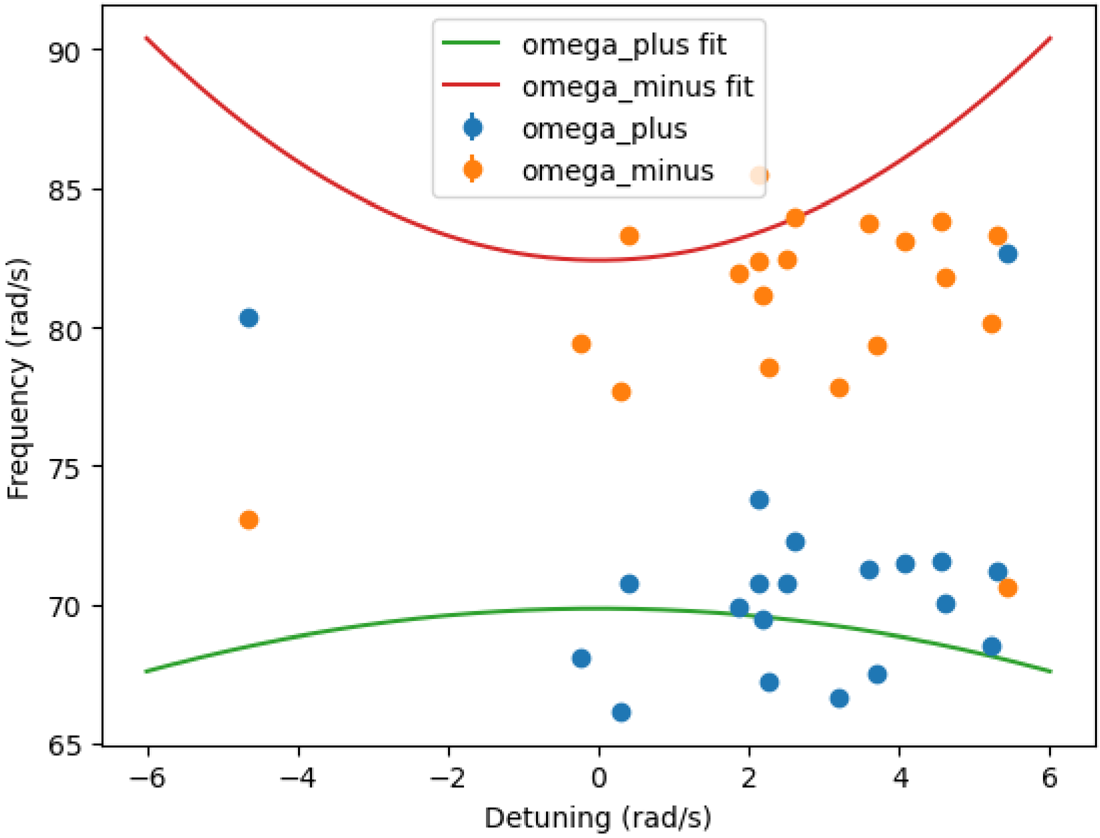
In [ ]:
```
print(fit_params1_u)
```

```
[[5.2831908608464815+/-0.01267016390940623, 0.5637400979881095+/-0.0010325
383338358946], [3.187684308313301+/-0.013185240056478564, 0.71534592233750
95+/-0.0011656237230191907], [4.595355768537046+/-0.014833057600325812, 0.
6102819314153171+/-0.0012399527407925375], [5.226924904474802+/-0.01360221
0118298255, 0.548284168956419+/-0.0011516851266831528], [0.389715272645506
4+/-0.013467169104883383, 0.9681241133896051+/-0.0011015498689590886], [2.
5059324820130744+/-0.021478717951822338, 0.7849636619241145+/-0.0018470931
148938175], [4.068661941854386+/-0.014457452376371655, 0.648633774672168+/
-0.0012381201175586938], [2.6057465915411+/-0.012542342130810657, 0.775451
5739233369+/-0.0010784743021435151], [2.121971359044168+/-0.01485690034484
048, 0.8191592049066053+/-0.0012643496566472976], [4.546130844138357+/-0.0
12658220431993553, 0.6251526950979296+/-0.0010410808890022327], [3.5864319
920108128+/-0.017515645875469204, 0.7022906617688278+/-0.00148060425097960
9], [2.1380813419469025+/-0.014725856213509343, 0.8132655185151229+/-0.001
2985357810653662], [1.8614370207651885+/-0.015065618684535686, 0.841672077
4636364+/-0.0012531012720696064], [2.187856746423425+/-0.01291267697310114
4, 0.8088441629287284+/-0.001111020030237521], [-4.656726945050587+/-0.122
81974082618781, 1.6277478280330673+/-0.014066820577823505], [-0.2522172449
2618613+/-0.015219800200089425, 1.0227359021057174+/-0.001373218893922385
8], [3.709682526990622+/-0.010854872676031264, 0.6779029675833492+/-0.0009
369491096808034], [2.2711110494631783+/-0.014969194939064442, 0.7953993019
410621+/-0.0013454612066427058], [0.30243747880048777+/-0.0146084880091158
93, 0.9727797385269734+/-0.0013136767000468464], [5.440930776545025+/-0.01
2670835242660989, 0.5395658455349749+/-0.001062977710156991]]
```

In [ ]:

```python
# Plot of omega_plus and omega_minus with detuning with error bars

# Detuning is first element in fit_params1_u
# omega_plus and omega_minus are third and fourth elements in fit_params1

omega_plus_u = np.array([fit_params1[i][2].nominal_value for i in range(l
omega_minus_u = np.array([fit_params1[i][3].nominal_value for i in range(

omega_plus_u_err = np.array([fit_params1[i][2].std_dev for i in range(len
omega_minus_u_err = np.array([fit_params1[i][3].std_dev for i in range(le

x_nom = np.array([fit_params1_u[i][0].nominal_value for i in range(len(fi
x_err = np.array([fit_params1_u[i][0].std_dev for i in range(len(fit_para


# Define fit function
def parabolic1(x, a, b):
    return a * x**2 + b

def parabolic2(x, a, b):
    return -a * x**2 + b
# Fit parabola to data
popt1, pcov1 = curve_fit(parabolic2, x_nom, omega_plus_u, sigma=omega_plu
popt2, pcov2 = curve_fit(parabolic1, x_nom, omega_minus_u, sigma=omega_mi

# Plot data
plt.errorbar(x_nom, omega_plus_u, yerr=omega_plus_u_err, fmt='o', label='
plt.errorbar(x_nom, omega_minus_u, yerr=omega_minus_u_err, fmt='o', label

# Plot fit
x = np.linspace(-6, 6, 1000)
plt.plot(x, parabolic1(x, *popt1), label='omega_plus fit')
plt.plot(x, parabolic2(x, *popt2), label='omega_minus fit')

plt.xlabel('Detuning (rad/s)')
```

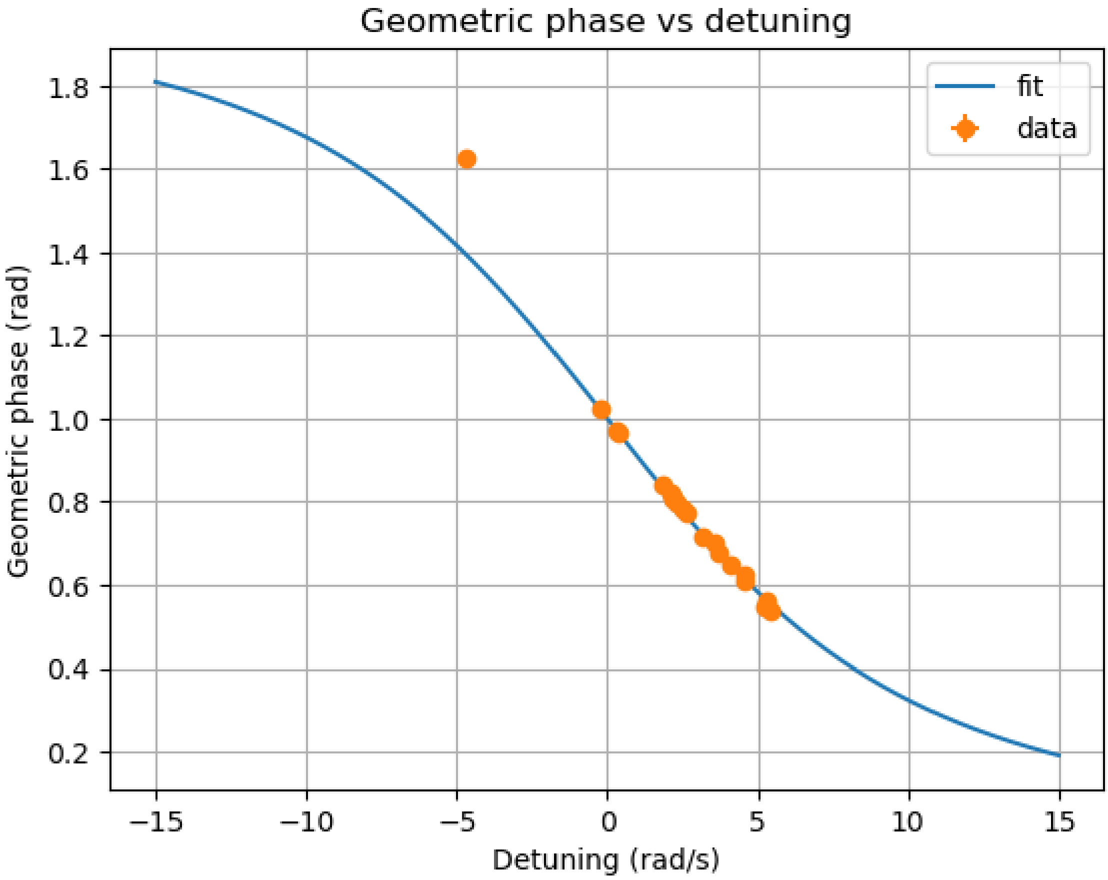
```python
plt.ylabel('Frequency (rad/s)')
plt.legend()
plt.savefig(f"{file1[-6:-4]}_freq_det.png",dpi=600)
plt.show()
```

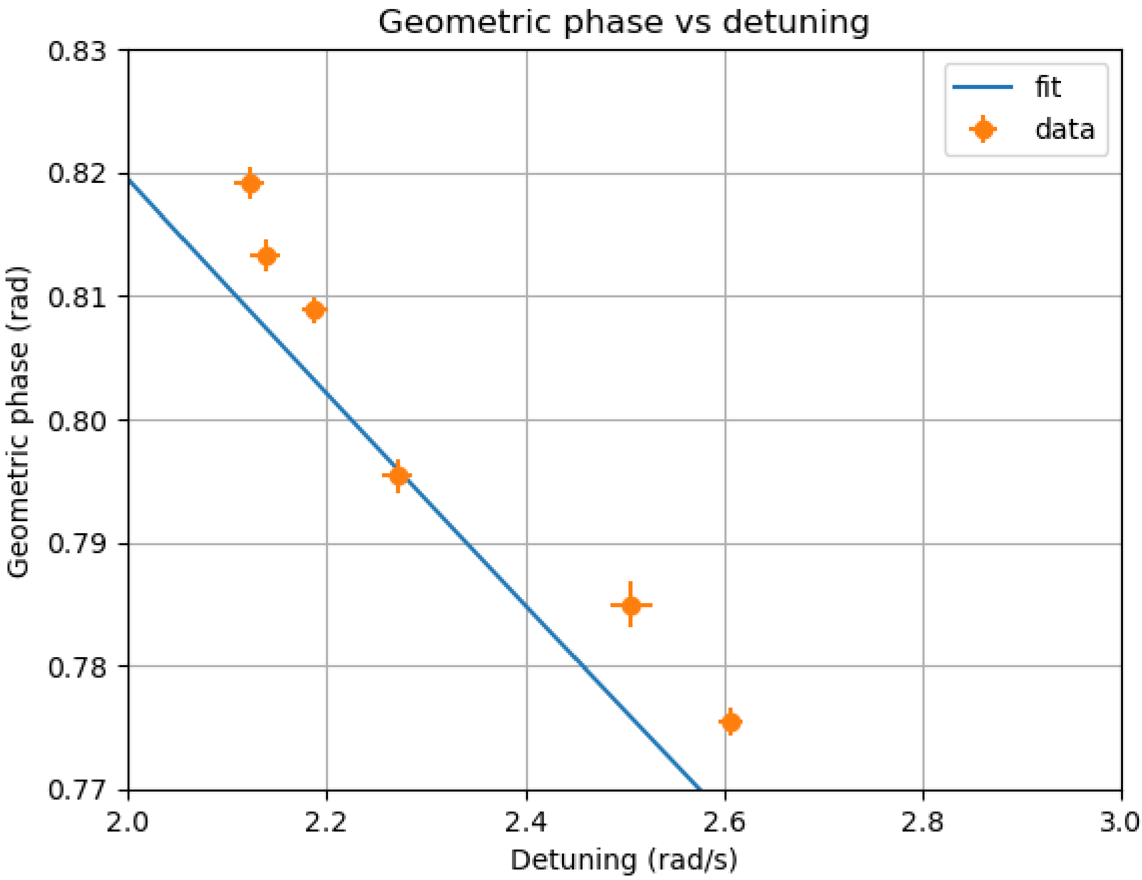


In [ ]:
```python
import matplotlib.pyplot as plt
from uncertainties import ufloat

# Geometric phase fit function
def phi_G(DELTA, OMEGA):
    return (1 - (DELTA / (np.sqrt(OMEGA**2 + DELTA**2))))

# Get x and y data points
x_nom = np.array([fit_params1_u[i][0].nominal_value for i in range(len(fi
x_err = np.array([fit_params1_u[i][0].std_dev for i in range(len(fit_para

y_nom = np.array([fit_params1_u[i][1].nominal_value for i in range(len(fi
y_err = np.array([fit_params1_u[i][1].std_dev for i in range(len(fit_para

print(x_nom)

# Plot data points with error bars



# Fit function
popt, pcov = curve_fit(phi_G, x_nom, y_nom, sigma=y_err, absolute_sigma=T

# Plot fitted function with figure size
x_fit = np.linspace(-15, 15, 1000)
y_fit = phi_G(x_fit, *popt)
plt.plot(x_fit, y_fit, label='fit')
plt.errorbar(x_nom, y_nom, xerr=x_err, yerr=y_err, fmt='o', label='data')
plt.title('Geometric phase vs detuning')
```

```
plt.xlabel('Detuning (rad/s)')
plt.ylabel('Geometric phase (rad)')
plt.legend()
plt.grid()
plt.savefig('Geometric_phase.png', dpi=600)
plt.show()

# Make another plot with x and y limits
plt.plot(x_fit, y_fit, label='fit')
plt.errorbar(x_nom, y_nom, xerr=x_err, yerr=y_err, fmt='o', label='data')
plt.title('Geometric phase vs detuning')
plt.xlabel('Detuning (rad/s)')
plt.ylabel('Geometric phase (rad)')
plt.legend()
plt.grid()
plt.xlim(2, 3)
plt.ylim(0.77, 0.83)
plt.savefig('Zoomed_in_geometric_phase.png', dpi=600)
plt.show()



# plt.xlabel('x')
# plt.ylabel('y')
# # plt.legend()
# plt.grid()
# plt.show()
```

```
[ 5.28319086  3.18768431  4.59535577  5.2269249   0.38971527  2.50593248
  4.06866194  2.60574659  2.12197136  4.54613084  3.58643199  2.13808134
  1.86143702  2.18785675 -4.65672695 -0.25221724  3.70968253  2.27111105
  0.30243748  5.44093078]
```




Geometric phase vs detuning
0.83
0.82
0.81
0.80
0.79
0.78
0.77
Geometric phase (rad)
2.0
2.2
2.4
2.6
2.8
3.0
Detuning (rad/s)
fit
data