PyLab - SpringMass

PHY224 Lab 3

Fredrik Dahl Bråten, Pankaj Patil

October 18, 2021

1 Abstract

In this exercise, we studied the equation of motion of a mass-spring system in damped and undamped cases. In both the cases the we verified the experimental data with that obtained using qualitative fit of equation of motion. This analysis was done in Python by use of the numpy, scipy and matplotlib modules.

2 Introduction

The undamped spring mass equation is given by the Hook's law,

$$F_{spring} = -ky$$

Where y is the vertical displacement of the mass and k is the spring constant. Above equation is used to derive the equation of motion of the spring mass system,

$$\frac{d^2y}{dt^2} + \Omega_0^2 y = 0 \implies y = y_0 + A\sin(\Omega_0 t) \tag{1}$$

Where $\Omega_0 = \sqrt{\frac{k}{m}},\ A = \text{Amplitude of Oscillations},\ y_0 = \text{Initial Position of the mass}$

In case of damping, we need to add damping force to the equation which depends on the velocity of the mass relative to surrounding medium. In our case, where we have small Reynolds numbers, the drag force is directly proportional to the velocity of the mass,

$$\vec{F_d} = -\gamma\vec{v}$$

Where γ is damping coefficient. The equation of motion is then given by

$$\frac{d^2y}{dt^2} + \gamma\frac{dy}{dt} + \Omega_0^2 y = 0 \implies y = y_0 + Ae^{-\gamma t}\sin(\Omega_0 t) \tag{2}$$

In both the cases, we qualitatively fit the data to the above displacement equations.

3 Methods, Materials and Experimental Procedure

We successfully followed the procedures as described by the TA for this experiment.

4 Results

4.1 Undamped Spring Mass

In Appendix Figure 2, we see the displacement data is plotted against time for the undamped case. The graph is a qualitative fit of the equation describing the undamped oscillatory motion of spring-mass system.

Using the qualitative fit we obtained following values,

y_0 = Initial Position = 20.71 cm

A = Amplitude = 0.70 cm

Ω_0 = Frequency of Oscillations = 9.07 rad/s

$T = 1/\Omega_0$ = Period of Oscillations = 0.110 rad/s

Using the Ω_0 value we obtain the spring constant

$$k = m\Omega_0^2 = 16.44\ kg/s^2$$

The simulated data for the undamped case is plotted in Figure 4.

4.2 Damped Spring Mass

In Appendix Figure 5, we see the displacement data is plotted against time for the damped case. The graph is a qualitative fit of the equation describing the damped oscillatory motion of spring-mass system.

Using the qualitative fit we obtained following values,

y_0 = Initial Position = 19.94 cm

A = Amplitude = 1.64 cm

γ = Damping Coefficient = 0.01

Ω_0 = Frequency of Oscillations = 8.69 rad/s

Using the Ω_0 value we obtain the spring constant

$$k = m\Omega_0^2 = 16.24\ kg/s^2$$

The simulated data for the damped case is plotted in Figure **??**.

5 Discussion

For the undamped spring-mass system,

$$
\begin{aligned}
&F_{spring} = -ky \qquad \text{Hook's Law}\\
&F = ma \qquad\qquad \text{Newton's Second Law of Motion}\\
&\implies ma = -ky\\
&\implies m\tfrac{d^2y}{dt^2} = -ky\\
&\implies m\tfrac{d^2y}{dt^2} + ky = 0
\end{aligned}
$$

We approximated that the motion of the system is purely one dimensional in only y-direction.

The equation of motion can be written as,

$$
\begin{aligned}
&\frac{d^2y}{dt^2} = -\Omega_0^2 y \quad \text{where } \Omega_0 = \sqrt{\frac{k}{m}}\\
\implies &\frac{dv}{dy} = -\Omega_0^2 y \quad \text{where } \frac{d^2y}{dt^2} = \frac{dv}{dy}, \quad v = \frac{dy}{dt}\\
\implies &\frac{1}{\Delta t}[v(t+\Delta t) - v(t)] = -\Omega_0^2 y \qquad \text{using Forward Euler method}\\
\implies &[v(t+\Delta t) - v(t)] = -\Delta t\Omega_0^2 y\\
\implies &v(t+\Delta t) = v(t) - \Delta t\Omega_0^2 y
\end{aligned}
$$

And

$$
\begin{aligned}
&v = \frac{dy}{dt}\\
\implies &\frac{1}{\Delta t}[y(t+\Delta t) - y(t)] = v(t) \qquad \text{using Forward Euler method}\\
\implies &y(t+\Delta t) = y(t) + \Delta t v(t)
\end{aligned}
$$

Thus the Forward Euler methods gives us,

$$
\begin{aligned}
y_{i+1} &= y_i + \Delta t v_i\\
v_{i+1} &= v_i - \Delta t\Omega_0^2 y_i
\end{aligned}
$$

for $i = 0, 1, 2, \ldots$

Above equations were used to compute the simulated oscillation which are plotted in Figure 4.

The oscillatory motion of spring mass system is a sinusoidal graph, which is expected as it is a periodic motion.

Our specified parameters produce a curve which fit our data very well.

In the Distance and Velocity vs. Time plots, we see that the Euler-Cromer simulation fit our data over time much better than the Forward Euler simulation, which amplitude grows significantly in time. The amplitude of the plots should however be constant, if not be weakly decreasing, due to the small unavoidable damping of our system of experiment, as seen in the plot of measured data.

As expected, we get elliptical phase plots for our measured and simulated Distance and Velocity, however, as we will see in the energy plot, the Forward Euler simulation grows in energy, which causes the phase space plot of the Forward Euler simulation ellipse to increase in radius over time. This also corresponds to the increase in amplitude in the position and velocity plot over time.

Furthermore, as we can see from the total Energy plots, the total energy of the system is approximately conserved for the Euler-Cromer simulation, grows in time and is not conserved in the Forward Euler simulation (which is unphysical for our system), and is approximately conserved for our measured data when accounting for uncertainties in our measurements of the distance over time. In theory, total energy should be conserved, but the energy will oscillate to be in the form of potential and kinetic energy.

The radius of the elliptical phase plots, correspond to the total energy of the system. When considering our total energy plots over time, it makes sense that the Euler Cromer and measured data plots give approximately stable phase ellipses, though the Forward Euler phase plot has increasing radius with time, as its energy is increasing over time. The reason the radius should be constant, is that the total energy should be constant. The y and v component of the phase plot radius oscillate in their contribution to the radius length, corresponding to how the total energy is conserved, but oscillates in the form of kinetic and potential energy.

For the undamped spring-mass system, the mechanical energy is conserved and is given by,

$$E_{tot} = \frac{1}{2}mv^2 + \frac{1}{2}ky^2$$

Rearranging above give,

$$\frac{v^2}{k} + \frac{y^2}{m} = \frac{2E_{tot}}{mk}$$

Which is an equation of an ellipse. Hence the phase plot of system is an ellipse.

In damped oscillations, as expected the amplitude of oscillation decays exponentially as is evident from Figure 5.

6 Conclusions

By approximating the motion in one dimension, we established that the equation of motion of spring-mass system in undamped case is given by Eqn (1). And that for damped system is give by Eqn. (2). In undamped system the energy of the system is conserved. Qualitative fit of the solution to equation of motion enables us to compute the spring constant, in both the cases and it is found to be in agreement within experimental errors. The qualitative fit establishes that the motion of spring-mass system is sinusoidal with constant period. In case of damping the amplitude of the motion decays exponentially. Furthermore, our two simulations using the Forward Euler and the Euler-Cromer methods, show that the Forward Euler method does not conserve energy, as the Euler-Cromer method approximately does. Therefore, the Euler-Cromer method is a better method for solving differential equations such as those we have worked with here.

A Appendix

A.1 Plots For Undamped Spring-Mass System

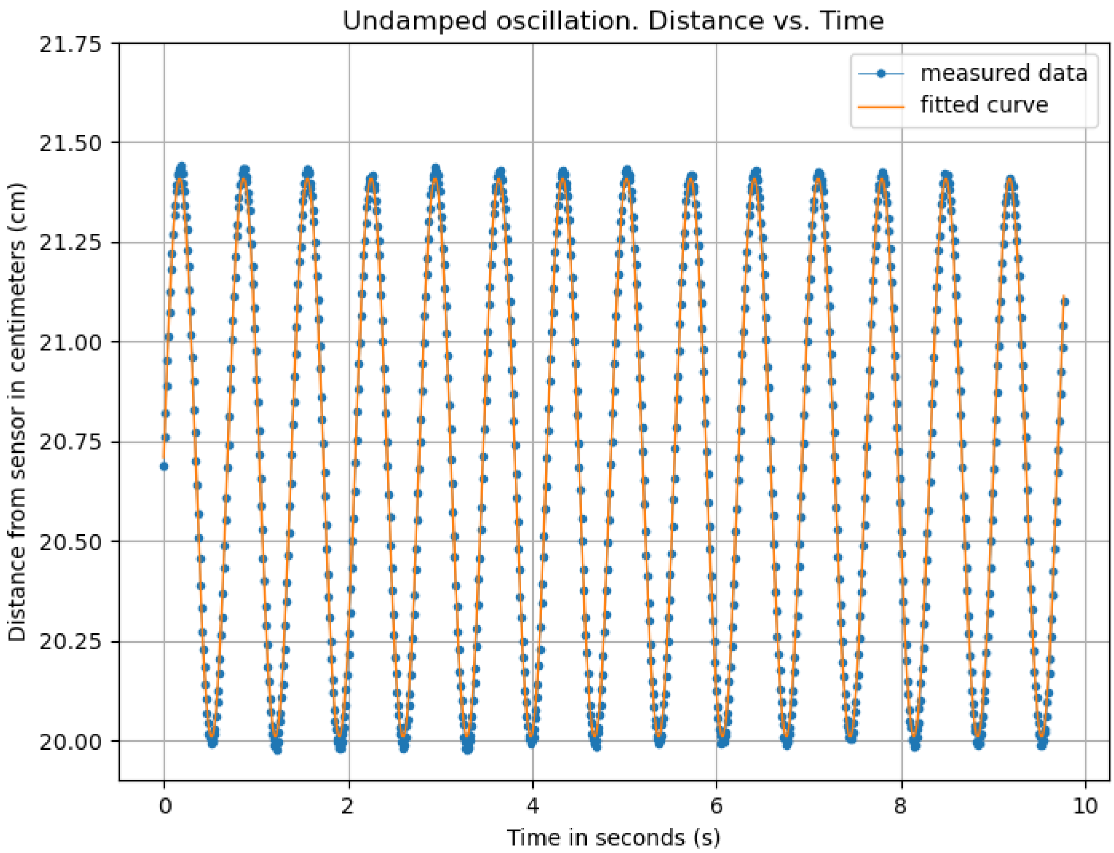

Figure 1: Undamped Oscillations: Distance vs. Time

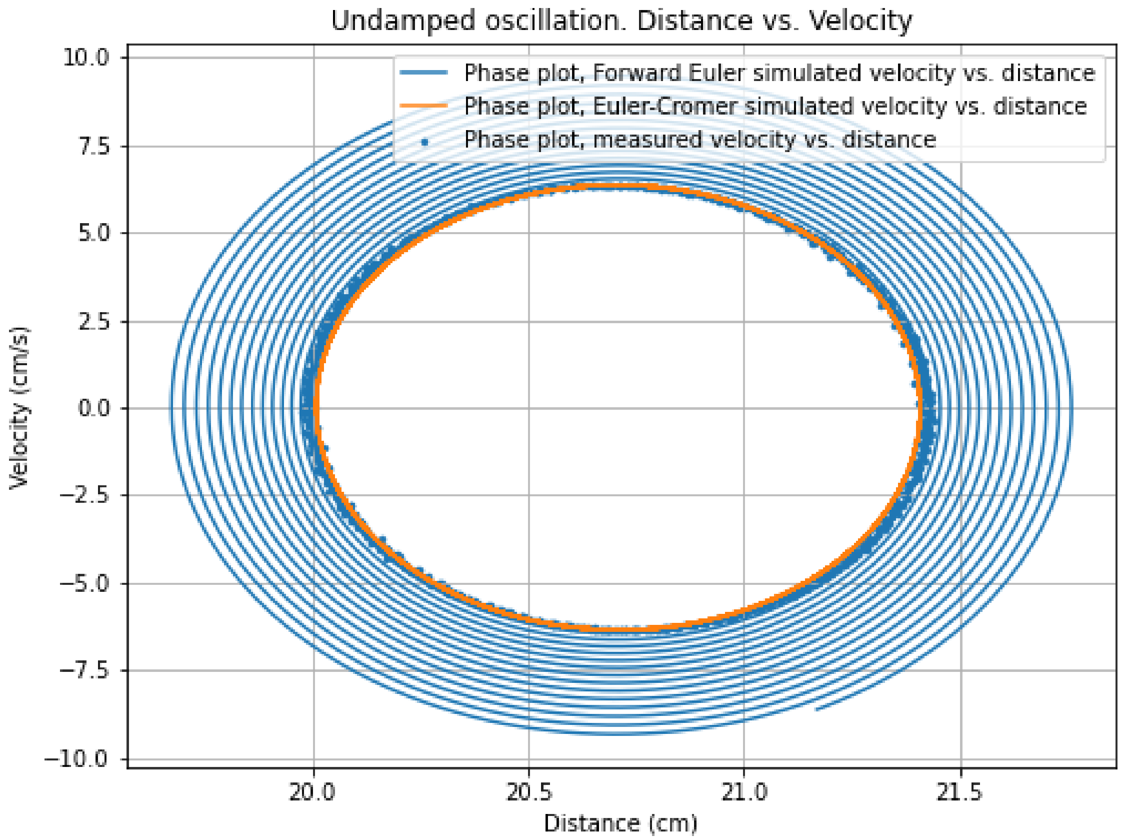

Figure 2: Undamped Oscillations: Velocity vs. Distance

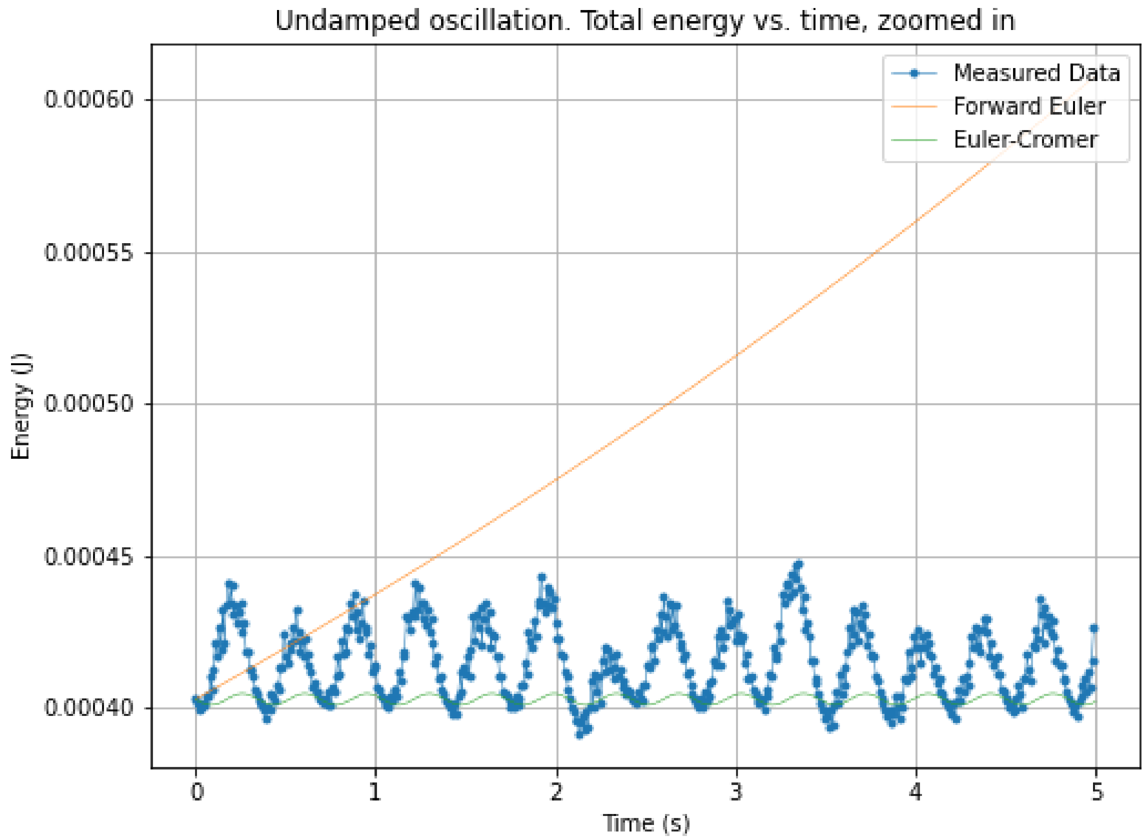

Figure 3: Undamped Oscillations: Total Energy vs. Time (zoomed in)

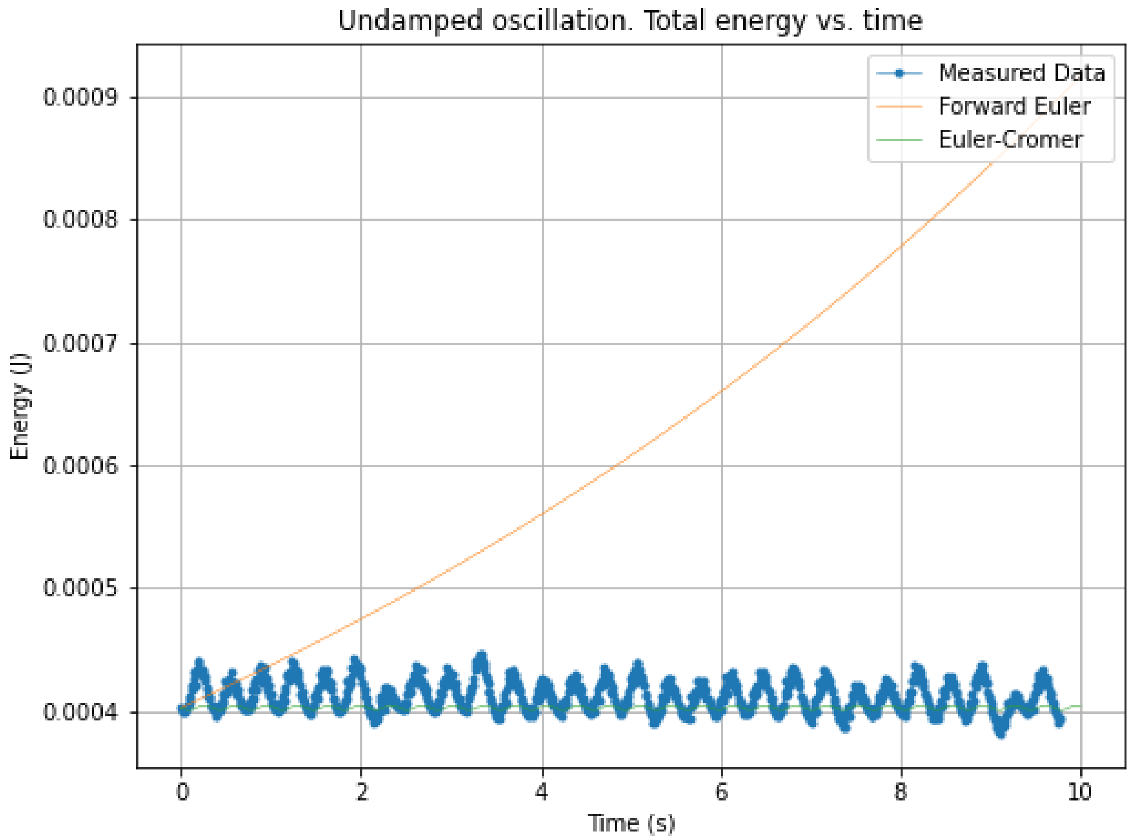

Figure 4: Undamped Oscillations: Total Energy vs. Time

A.2 Plots For Damped Spring-Mass System

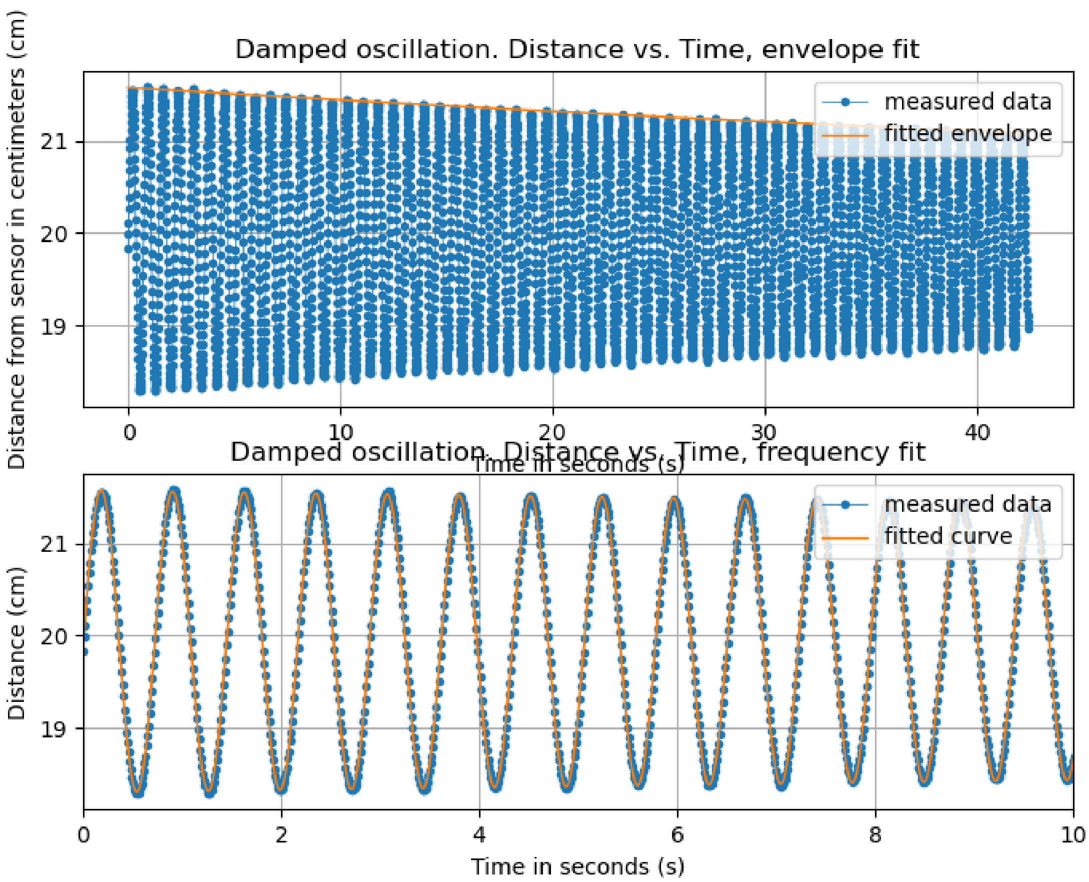

Figure 5: Damped Oscillations

A.3 Python Code

The Python code for this exercise is divided into two files. Functions.py file contains utility methods which we will be frequently using in this course. Undamped_and_Damped.py file contains the code which analyzes the data.

A.3.1 Functions.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 30 09:46:36 2021

@author: Fredrik
"""
import numpy as np
import scipy.optimize as optim
import matplotlib.pyplot as plt

#Defining the function for curve fitting and plotting
def curve_fit_and_plot(model,initial_guess,xdata,ydata,y_uncer,xunit,yunit,
                       plot_title):
    """
    This function uses the scipy curve_fit function to estimate the parameters
    of the model which will minimize the euclidian distance between our data
    points, and the model curve.
    We print these optimal model parameters along with their uncertainty, and
    plot the original data with error bars, along with the curve fit model.

    Parameters
    ----------
    model : function to be used as model
        model(x,a,b,c,...), where we are estimating a,b,c, etc.
    initial_guess : list of guesses for the parameters a,b,c, etc. eg. [2,4,254]
    xdata : list of input points for the model, eg. [2,4,5,7,9,28]
    ydata : list of output points for the model, eg [23,25,26,85,95,104]
    y_uncer : list of uncertaintes associated with the ydata, which the model
        shall output
    xunit : String describing the unit along the x-axis for label when plotting.
        eg. 'Voltage (V)'
    yunit : String describing the unit along the y-axis for label when plotting.
        eg. 'Current (A)'
    plot_title : String describing the title of the plot.
        eg. 'Current vs. Voltage'
```

```
    Returns None
    """
    #Using the scipy curve fit function to find our model parameters
    p_opt , p_cov = optim.curve_fit(model , xdata , ydata, p0 = initial_guess,
                                    sigma = y_uncer, absolute_sigma = True )
    p_std = np.sqrt( np.diag ( p_cov ))

    print("The optimal values for our curve fit model parameters, are:",np.round(p_opt,2))
    print("Their associated uncertainties  are:", np.round(p_std,2))

    #Now we create some data points on the model curve for plotting
    xvalues_for_plot = np.linspace(xdata[0],xdata[-1],1000)
    yvalues_for_plot = []
    for i in xvalues_for_plot:
        yvalues_for_plot.append(model(i,p_opt[0],p_opt[1]))

    #Now we plot the original data with error bars, along with the curve fit model
    plt.figure(figsize=(10,5))
    plt.errorbar(xdata,ydata,y_uncer,c='r', ls='', marker='o',lw=1,capsize=2,
                 label = 'Points of measurement with uncertainty')
    plt.plot(xvalues_for_plot,yvalues_for_plot, c='b',
             label = 'Scipy curve fit')
    plt.title(plot_title)
    plt.xlabel(xunit)
    plt.ylabel(yunit)
    plt.legend()
    plt.grid()
    plt.savefig(plot_title+'.png')
    plt.show()

    return None

def error_plot(model,p_opt,xdata,ydata,y_uncer,xunit,yunit,
                   plot_title):
    #Now we create some data points on the model curve for plotting
    xvalues_for_plot = np.linspace(xdata[0],xdata[-1],1000)
    yvalues_for_plot = []
    for i in xvalues_for_plot:
        yvalues_for_plot.append(model(i,p_opt[0],p_opt[1]))

    #Now we plot the original data with error bars, along with the curve fit model
    plt.figure(figsize=(10,5))
    plt.errorbar(xdata,ydata,y_uncer,c='r', ls='', marker='o',lw=1,capsize=2,
                 label = 'Points of measurement with uncertainty')
    plt.plot(xvalues_for_plot,yvalues_for_plot, c='b',
             label = 'Scipy curve fit')
```

```
    plt.title(plot_title)
    plt.xlabel(xunit)
    plt.ylabel(yunit)
    plt.legend()
    plt.grid()
    plt.savefig(plot_title+'.png')
    plt.show()
    return None

def chi2(y_measure,y_predict,errors):
    """Calculate the chi squared value given a measurement with errors and
    prediction"""
    return np.sum( np.power(y_measure - y_predict, 2) / np.power(errors, 2) )

def chi2reduced(y_measure, y_predict, errors, number_of_parameters):
    """Calculate the reduced chi squared value given a measurement with errors
    and prediction, and knowing the number of parameters in the model."""
    return chi2(y_measure, y_predict, errors)/ \
            (y_measure.size - number_of_parameters)

def read_data(filename, Del, skiprows, usecols=(0,1)):
    """Load give\n file as csv with given parameters,
    returns the unpacked values"""
    return np.loadtxt(filename,
                      skiprows=skiprows,
                      usecols=usecols,
                      delimiter=Del,
                      unpack=True)

def fit_data(model_func, xdata, ydata, yerrors, guess):
    """Utility function to call curve_fit given x and y data with errors"""
    popt, pcov = optim.curve_fit(model_func,
                                 xdata,
                                 ydata,
                                 absolute_sigma=True,
                                 sigma=yerrors,
                                 p0=guess)

    pstd = np.sqrt(np.diag(pcov))
    return popt, pstd
```

A.3.2 Functions.py

```
#Importing modules
import numpy as np
import matplotlib.pyplot as plt
import Functions as F


##Undamped Oscillation:

#Specifying modelling function
def model(t,a,b,c):
    return a+b*np.sin(c*t)

#Importing data
Time, Distance = F.read_data('undamped_point_data_set2.txt', None,2)

#Defining Constants
Sample_time = 0.01 #seconds
m = 200.0/1000 #Kilograms
m_uncertainty = 0.1/1000 #kilograms

#Specifying parameters for the model function
a = np.mean(Distance)
b = 0.7
c = (2*np.pi)/0.693

print("Initial Position = %.2f" % a)
print("Amplitude = %.2f" % b)
print("Frequency = %.2f" % c)

#Calculating the spring constant based on these parameters
spring_constant = m * c**2
print("The spring constant of the string estimated in the undamped system",
      "exercise, is:", spring_constant, "kg/s^2")

#Offsetting time array
Time = np.array([i-0.49 for i in Time])

#Plotting data points and model curve
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(1,1,1)
ax.plot(Time, Distance, marker='.',lw=0.5,label='measured data')
ax.plot(Time, model(Time,a,b,c),lw=1,label='fitted curve')
ax.set_ylim((19.9, 21.75))
```

```
ax.legend(loc=1)
ax.set_xlabel("Time in seconds (s)")
ax.set_ylabel("Distance from sensor in centimeters (cm)")
ax.set_title("Undamped oscillation. Distance vs. Time")
ax.grid()
ax.figure.savefig("Undamped oscillation. Distance vs. Time"+".png")
plt.show()

##Damped Oscillation:


#Specifying modelling function
def model(t,a,b,c,d):
    return a+b*np.exp(-c*t)*np.sin(d*t)

def env(t,a,b,c):
    return a+b*np.exp(-c*t)

#Importing data
Time, Distance = F.read_data('damped_point_data.txt', None,2)

#Defining Constants
Sample_time = 0.01 #seconds
m = 215.1/1000 #Kilograms
m_uncertainty = 0.1/1000 #kilograms

#Specifying parameters for the model function
a = np.mean(Distance)
b = 1.64
c = 0.0085
d = (2*np.pi)/0.723

print("Initial Position = %.2f" % a)
print("Amplitude = %.2f" % b)
print("Damping Coefficient = %.2f" % c)
print("Frequency = %.2f" % d)

#Calculating the spring constant based on these parameters
spring_constant = m * d**2
print("The spring constant of the string estimated in the damped system",
      "exercise, is:", spring_constant, "kg/s^2")

#Offsetting time array
Time = np.array([i-6.630 for i in Time])#[4000:-1]
Distance = Distance#[4000:-1]

#Plotting data points and model curve
```

```
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(2,1,1)
ax.plot(Time, Distance, marker='.',lw=0.5,label='measured data')
ax.plot(Time, env(Time,a,b,c),lw=1,label='fitted envelope')
#ax.set_ylim((19.9, 21.75))
ax.legend(loc=1)
ax.set_xlabel("Time in seconds (s)")
ax.set_ylabel("Distance from sensor in centimeters (cm)")
ax.set_title("Damped oscillation. Distance vs. Time, envelope fit")
ax.grid()
ax.figure.savefig("Damped oscillation. Distance vs. Time"+".png")

ax = fig.add_subplot(2,1,2)
ax.plot(Time, Distance, marker='.',lw=0.5,label='measured data')
ax.legend(loc=1)
ax.set_title("Damped oscillation. Distance vs. Time, frequency fit")

ax.plot(Time, model(Time,a,b,c,d),lw=1,label='fitted curve')
#ax.set_ylim((19.9, 21.75))
ax.legend(loc=1)
ax.set_xlabel("Time in seconds (s)")
ax.set_ylabel("Distance (cm)")
ax.set_xlim(0,10)
ax.grid()
ax.figure.savefig("Damped oscillation. Distance vs. Time"+".png")
plt.show()



## simulation:
```

References

[1] Lab Manual - Spring Mass - exercise4_NI.pdf