
Wave Phenomena Lab 6

Python Simulation for Coupled Oscillators

Mahathi Narayanaswamy

Instructor: Proteep Malik

Academic Associate: Sameer Behera

Date Performed: 19 November 2021

Submitted on: 5 December 2021

Abstract

A computer simulation of coupled oscillators is carried out using Python programming in order to study their behavior. In the first part of the simulation we attempt to calculate the normal frequencies of a two-mass coupled oscillator by solving a set of differential equations using matrix method, thus equating the determinant to zero. In the second part, we try to plot the position, velocity and acceleration of a one-mass oscillator. In the third part we try to simulate the motion of a n-coupled oscillator and lastly in the fourth part we try to obtain lissajous figures. As an extension of the third part we also fit the motion of 2-mass and 3-mass oscillators to their theoretical curves and calculate their normal frequencies which are represented in Table 2. The obtained results are inserted in the form of figures and tables in the report. All the obtained results were in accordance with theoretical predictions with the n-coupled oscillators fitting to the equation

$$x = A_1 cos(w_A t + \delta_1) + A_2 cos(w_B t + \delta_2) + A_3 cos(w_C t + \delta_3).$$

Python Simulation for Coupled Oscillators

Codes

Disclaimer: Code numbers in this report do not correspond to those in the lab instruction sheet.

Code 1: Normal frequencies of 2-coupled systems

This piece of code asks the user to input the values of the mass and the spring constant of the two-mass coupled system and calculates the resultant normal frequencies.

```
#necessary modules
import numpy as np

#inputdata
m = float(input("Enter a positive number for the mass"))
k = float(input("Enter a positive number for the stiffness"))

for w in np.arange(-100, 100, 0.01):
    mat = np.array([[(-m*(w**2)+(2*k)),-k],[-k, (-m*(w**2)+(2*k))]])
    if -0.01 <= np.linalg.det(mat) <= 0.01:
        print(mat)
        print(w, np.linalg.det(mat))
```

Code 2: Displacement, Velocity and Acceleration of 1-mass coupled oscillator

This piece of code plots the displacement vs time, velocity vs time and acceleration vs time graphs for a 1-mass coupled oscillator by asking the user to input the initial displacements given to the mass.

```
#importing necessary modules
import numpy as np
import matplotlib.pyplot as plt

#definitions
#DEFINITION 1: generating an n by n symmetric matrix with diagonal
elements = -2, the neighbouring elements = 1
```

```
def matrix(n):
        Matrix = np.zeros((n, n))
        for i in range(n):
            for j in range(n):
                if i == j:
                    Matrix[i,j] = -2
                    if i != 0:
                        Matrix[i-1,j] = 1
                    if i != n-1:
                        Matrix[i+1,j] = 1
        return Matrix

#DEFINITION 2: updating matrix
def n_coupled(x0, w, dt = 0.01, total_time = 20):
    positionoutput = {}
    V = np.zeros(len(x0), dtype='float64')
    Matrix = matrix(len(x0)).astype('float64')
    X = x0.astype('float64')
    for t in np.arange(0,total_time+dt,dt):
        A = np.matmul(w*Matrix,X) #equation of motion
        V += A*dt #updating position and velocity
        X += V*dt
        positionoutput[t] = np.copy(X)
    return positionoutput

w = 1
dt = 0.01
X0 = np.array([1])
output = n_coupled(X0, w)
position = np.array([output[t][0] for t in output.keys()])
velocity = np.delete((np.roll(position, 1)-position)/dt, 0)
acceleration = np.delete((np.roll(velocity, 1)-velocity)/dt, 0)

#plotting
plt.plot(list(output.keys())[:], position)
plt.title("Position")
plt.xlabel("Time(arbitrary units)")
plt.ylabel("Position(arbitrary units)")
```

```
plt.plot(list(output.keys())[:-1], velocity)
plt.title("Velocity")
plt.xlabel("Time(arbitrary units)")
plt.ylabel("Velocity(arbitrary units)")

plt.plot(list(output.keys())[:-2], acceleration)
plt.title("Acceleration")
plt.xlabel("Time(arbitrary units)")
plt.ylabel("Acceleration(arbitrary units)")
```

Code 3: Displacements of n-coupled Oscillators

This piece of code asks the user to input the value of n,the angular frequency and the initial displacements of each of n-springs. It outputs a position vs time graph of the n oscillators. This

```
#importing necessary modules
import numpy as np
import matplotlib.pyplot as plt

#definitions
#DEFINITION 1: generating an n by n symmetric matrix with diagonal
elements = -2, the neighbouring elements = 1
def matrix(n):
        Matrix = np.zeros((n, n))
        for i in range(n):
            for j in range(n):
                if i == j:
                    Matrix[i,j] = -2
                    if i != 0:
                        Matrix[i-1,j] = 1
                    if i != n-1:
                        Matrix[i+1,j] = 1
        return Matrix

#DEFINITION 2: updating matrix
def n_coupled(x0, w, dt = 0.01, total_time = 20):
    positionoutput = {}
```

```
    V = np.zeros(len(x0), dtype='float64')
    Matrix = matrix(len(x0)).astype('float64')
    X = x0.astype('float64')
    for t in np.arange(0,total_time+dt,dt):
        A = np.matmul(w*Matrix,X) #equation of motion
        V += A*dt #updating position and velocity
        X += V*dt
        positionoutput[t] = np.copy(X)
    return positionoutput

#input data:
n = int(input("Enter the number of oscillators"))
w = float(input("Enter a positive number for the angular frequency:
note that this is root(k/m) and thus takes into account both the mass
and stiffness"))
if w <= 0:
    print("angular frequency has to be a positive number")
    exit
initialpositions = []
for i in range(0,n):
    e = float(input("input the initial displacement of the ith
mass"))
    initialpositions.append(e)
x0 = np.array(initialpositions)

#n-coupled code
s = n_coupled(x0, w)
m = []
for i in range(0,n):
    m.append([s[t][i] for t in s])

#plotting
plt.xlabel("Time(arbitrary units)")
plt.ylabel("Position(arbitrary units)")

for i in range(0,n):
    plt.plot(s.keys(), m[i], label = "Mass"+str(i+1))

plt.legend(loc='upper right', bbox_to_anchor=(1.3, 1))
```

```
plt.tight_layout()
#plt.savefig('fig.png')
```

Code 4: Lissajous Figures

This code runs for a system of 4 springs that are connected to a single mass, with two connected along horizontal direction and two connected along vertical direction. It asks the user to input the displacement given to the object along the x and y directions and also asks the user to input the stiffness of the springs along the x and y directions. It outputs the graph of the resulting lissajous figure for that combination of stiffness.

```
#importing necessary modules
import numpy as np
import matplotlib.pyplot as plt

#definitions
#DEFINITION 1: generating an n by n symmetric matrix with diagonal
elements = -2, the neighbouring elements = 1
def matrix(n):
        Matrix = np.zeros((n, n))
        for i in range(n):
            for j in range(n):
                if i == j:
                    Matrix[i,j] = -2
                    if i != 0:
                        Matrix[i-1,j] = 1
                    if i != n-1:
                        Matrix[i+1,j] = 1
        return Matrix

#DEFINITION 2: updating matrix
def n_coupled(x0, w, dt = 0.01, total_time = 10):
    positionoutput = {}
    V = np.zeros(len(x0), dtype='float64')
    Matrix = matrix(len(x0)).astype('float64')
```

```
    X = x0.astype('float64')
    for t in np.arange(0,total_time+dt,dt):
        A = np.matmul(w*Matrix,X) #equation of motion
        V += A*dt #updating position and velocity
        X += V*dt
        positionoutput[t] = np.copy(X)
    return positionoutput

#input data
X0 = float(input("input the initial displacement of the mass in X
direction"))
Y0 = float(input("input the initial displacement of the mass in Y
direction"))
w1 = float(input("input the angular frequency of springs along X
direction"))
w2 = float(input("input the angular frequency of springs along Y
direction"))

x0 = np.array([X0])
y0 = np.array([Y0])

dt = 0.01
results_x = n_coupled(x0, w1)
results_y = n_coupled(y0, w2)

plt.plot(results_x.values(), results_y.values())
plt.xlabel("x-position (arbitrary units)")
plt.ylabel("y-position (arbitrary units)")
plt.title("Lissajous Figures")
```

Output Data and Data Objects from Code

Code 1: Normal frequencies of 2 coupled-system

k(in N/m)	m(in kg)	w(in rad/sec)
8	2	2
7	2	3.24
5	2	1.58

Table 1: Represents resultant frequencies of 2-coupled systems

Code 2: Displacement, Velocity and Acceleration of 1-mass coupled oscillator

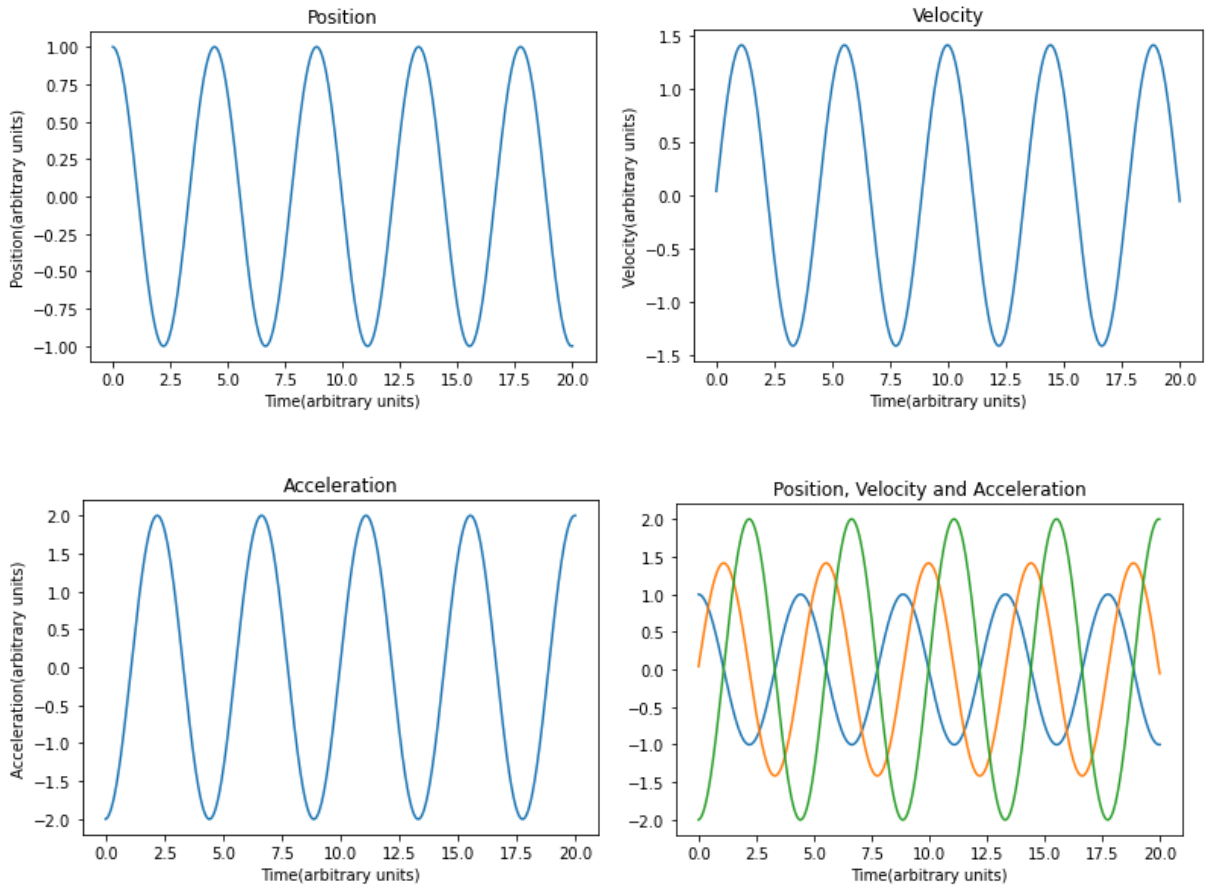

Figure 1-4: (1: displacement-time graph), (2: velocity-time graph), (3: acceleration-time graph), (4: graph showing position, velocity and acceleration vs time on the same graph with blue representing position, orange representing velocity and green representing acceleration)

The graphs of position, velocity and acceleration versus time as depicted in Figures 1-4 are as expected theoretically. Each graph is sinusoidal in nature. Each of the graphs is phase shifted from the previous by $\frac{\pi}{2}$, as expected by theory. The frequency of all the curves is the same.

Code 3: Displacements of n-coupled oscillators

2-mass system

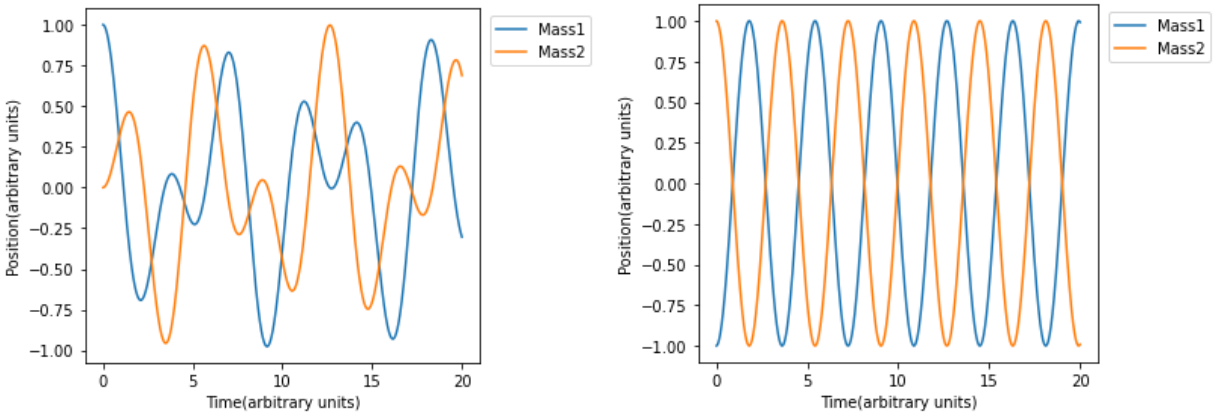

Figure 5-6: (5: k=1, One mass displaced), (6: k=1, Both masses displaced)

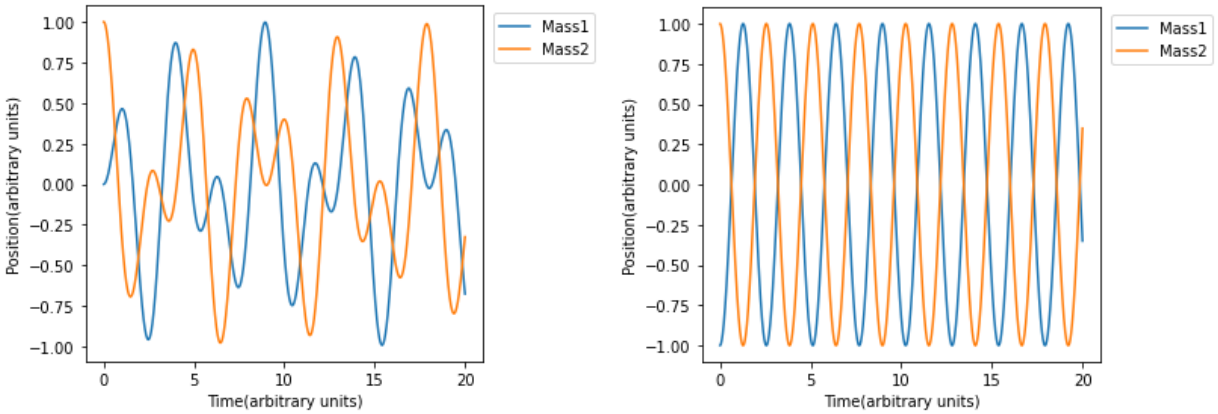

Figure 7-8: (7: k=2, One mass displaced), (8: k=2, Both masses displaced)

The graphs obtained in 5-8 are in accordance with theoretical predictions. The graphs that are produced as a result of increasing angular frequency, only results in an increase in the number of oscillations per unit time and the same can be observed by comparing Figures 5 and 7 and similarly Figures 6 and 8.

3-mass system

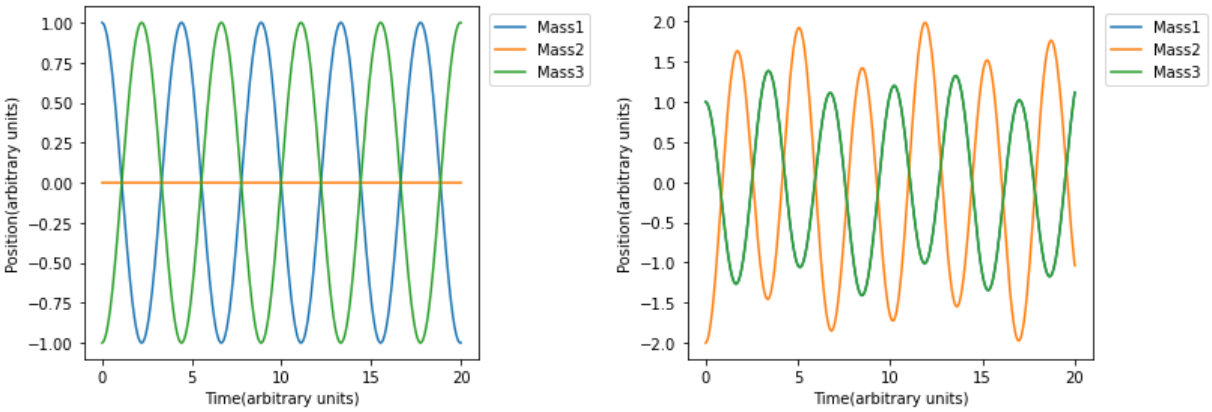

Figure 9-10: Two normal modes of 3-mass coupled systems

(9: w=1, x_1=-1,x_2=0 and x_3=1), (10: w=1, x_1=1,x_2=-2 and x_3=1)

10-mass system(just cause it looks cool)

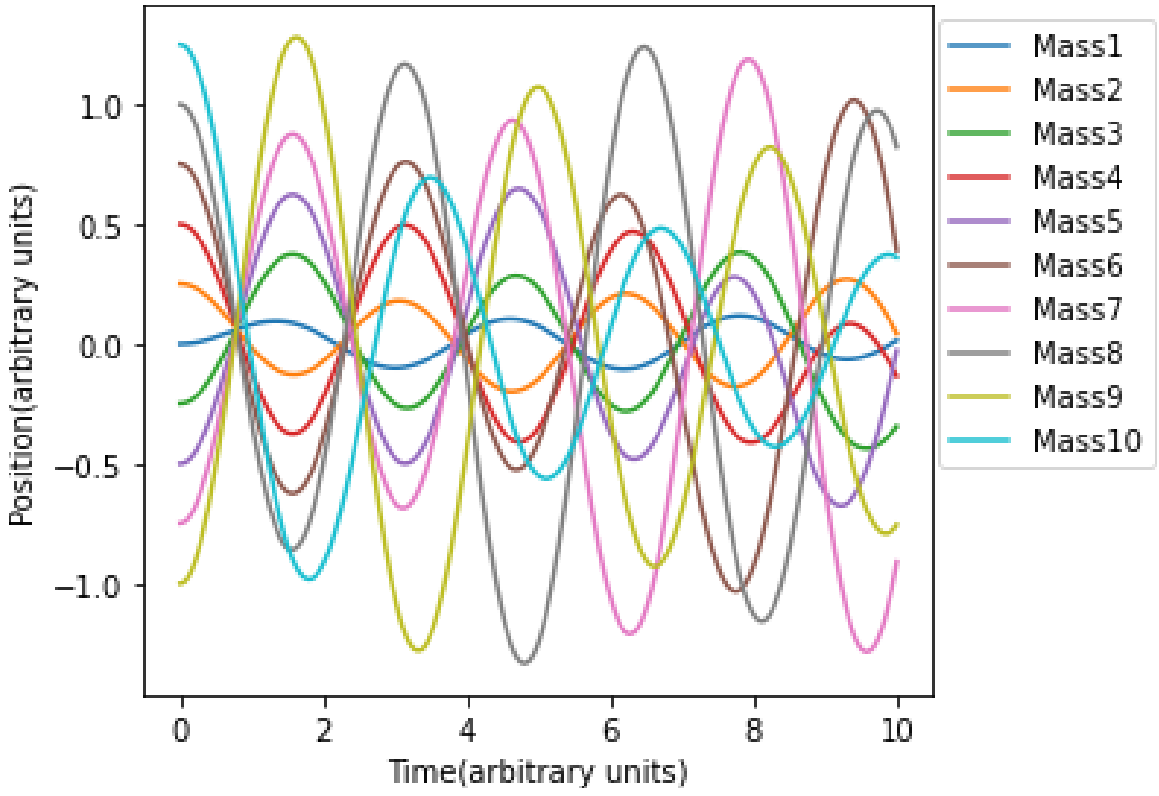

Figure 11: 10-mass system

Code 4: Lissajous Figures

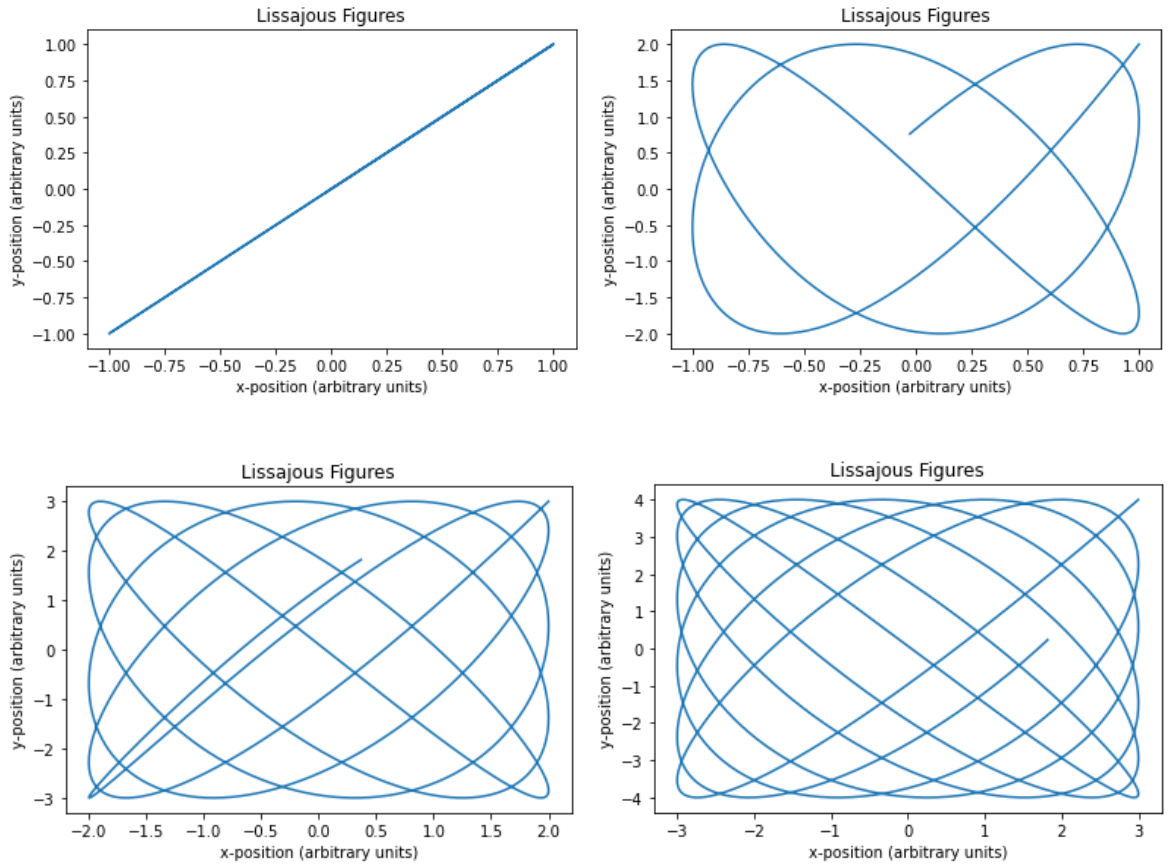

Figure 12-15: 12: Lissajous Figure due to 1:1 ratio, 13: Lissajous Figure due to 1:2 ratio, 14: Lissajous Figure due to 2:3 ratio, 15: Lissajous Figure due to 3:4 ratio

Lissajous figures obtained for various ratios are shown in Figures 12-15. The output images are as expected based on theory. The same images were also obtained during the Lissajous Figures lab.

Data Analysis and Curve Fitting

The data obtained from Code 3 for n=2 and n=3 with displacements of (1,0), (1,1), (1,0,-1) and (1,-2,1) were exported using the Pandas module into Excel(please refer spreadsheets). LibreOffice CalC was then used to fit them to their theoretical curves and solve for the resulting

coupling frequencies. Additionally, an attempt to discuss/explain any deviation from the theoretical curves is made in the following section on Errors.

For all coupled systems the value of w is set as 1. The data obtained from the curve fitting is as represented in Figures 16-19 and Table 2.

The theoretical curves they are fit to are of the form:

$$x = A_1 cos(w_A t + \delta_1) + A_2 cos(w_B t + \delta_2) + A_3 cos(w_C t + \delta_3).$$

For a two mass system $A_3 = 0$, as a result of which the last term is always 0.

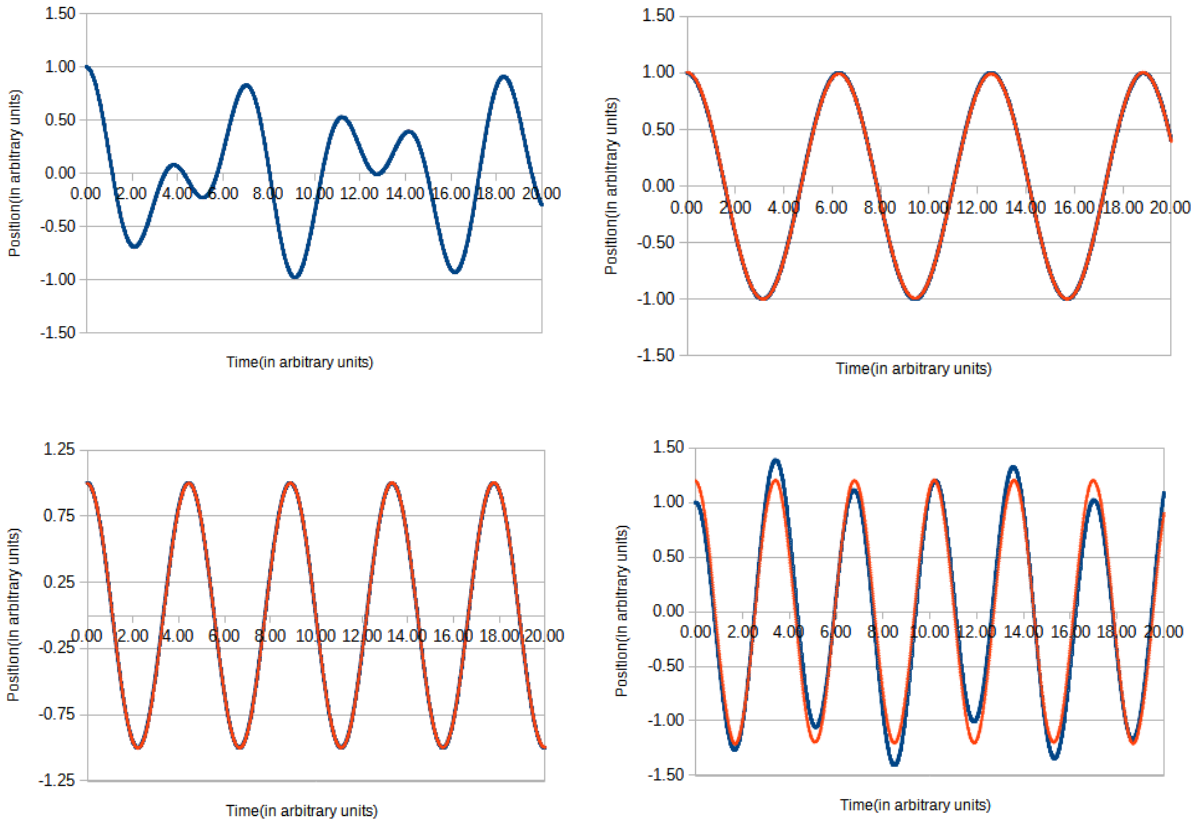

Figure 16-19: (16: position-time graph for two coupled oscillators with first mass displaced by 1 unit and second mass being displaced by 0 units), (17: position-time graph for two coupled oscillators with first mass and second mass being displaced by 1 unit each), (18: position-time graph for three coupled oscillators with first mass displaced by 1 unit, second mass being displaced by 0 units and third mass being displaced by -1 units), (19: position-time

graph for three coupled oscillators with first mass displaced by 1 unit, second mass being displaced by -2 units and third mass being displaced by 1 units)

Initial displacement x_A	Normal frequency w_A	Initial displacement x_B	Normal frequency w_B	Initial displacement x_C	Normal frequency w_C
1	1.00	0	1.73	-	-
1	1.00	1	1.18	-	-
1	1.41	0	0.00	-1	0.01
1	8.36	-2	1.85	1	8.36

Table 2: Resultant frequencies and the initial displacements for each of the four cases

Additionally, the beat and sum frequencies for the two-mass systems are calculated to be 0.73 and 2.73 for the first case and 0.18 and 2.18 for the second case using the formulae:

$w_{beat} = |w_A - w_B|$ and

$w_{sum} = w_A + w_B$

For the three mass oscillators we can calculate the sum frequency using:

$w_{sum} = w_A + w_B + w_C$

The same can be extended to n-mass oscillators. However, we cannot calculate a single beat frequency for $n \geq 2$ since the superposition of more than two waves are involved in those cases.

Errors

The error/deviation that is obtained from curve fitting is very small for 2-mass cases and the first 3-mass case. Any error/deviation is systematic and the same output data will be obtained no matter how many times we run the code provided we do not change the input data and the intervals(i.e. dt) since there is no source of random errors.

The lack of accurate fit for the 3 mass system is likely to be due to Tracker and not the simulation. The results obtained from the simulation seem accurate to theoretical predictions and are likely to be obtained as the results of an actual experiment under ideal conditions.

References

Azim Premji University. (2021). *Lab 6: Python Simulation for Coupled Oscillators*.