Solution Review: Harmonographs

Solution review to the harmonographs exercise.

WE'LL COVER THE FOLLOWING

Solution 1

```
import numpy as np
import matplotlib.pyplot as plt

# defining function with default paramters value
def pendulum(t, A=1, f=100, p=0, d=0.05):
  return (A * np.sin (2 * np.pi * f * t + p) * np.exp(-1 * d * t))

# creating array for time
t = np.linspace(0, 2 * np.pi, 1000)
x_t = pendulum(t, 1, 2, 0, 0.5)

# plotting and setting labels
plt.figure(figsize=(12, 9), dpi=200)
plt.plot(t, x_t)

# setting labels
plt.title('Damped Pendulum')
plt.xlabel('t')
plt.ylabel('x(t)')

# saving figure
plt.savefig('output/harmonograph1.png')
```

Explanation

- In lines 5 - 6, we have defined the function `pendulum()` and set the default values for `A`, `f`, `p`, and `d`.
- In line 9, we have created an array for time `t`.
- In line 10, we have used custom arguments for the `pendulum()` function to generate data for a pendulum moving along the x-axis.
- In lines 13 - 14, we are plotting the motion of the pendulum `x_t` against time `t`.

Solution 2

We have to plot the following equations:

$$x(t) = 2sin(8\pi t - 1)e^{-0.05t} + sin(4\pi t - \frac{\pi}{2})e^{-0.01t}$$

$$y(t) = 2sin(4\pi t - \frac{\pi}{3})e^{-0.02t} + 2sin(4\pi t - \pi)e^{-0.25t}$$

t is in the range $0 \leq t < 20\pi$ with $40,000$ values.

```
import numpy as np
import matplotlib.pyplot as plt

# defining function with default paramters value
def pendulum(t, A=1, f=100, p=0, d=0.05):
  return (A * np.sin (2 * np.pi * f * t + p) * np.exp(-1 * d * t))

# creating array for t
t = np.linspace(0, 20 * np.pi, 40000)

# creating x(t) and y(t) using the pendulum function
x_t = pendulum(t, 2, 4, -1, 0.05) + pendulum(t, 1, 2, np.pi/2, 0.01)
y_t = pendulum(t, 4, 2, np.pi/4, 0.02) + pendulum(t, 1, 2, np.pi, 0.25)

# plotting commands
plt.figure(figsize=(12, 9), dpi=200)
plt.plot(x_t, y_t, linewidth='0.25')

# setting limits for plots
plt.xlim(min(x_t)-0.5 , max(x_t)+0.5)
plt.ylim(min(y_t)-0.5,  max(y_t)+0.5)

# setting labels for plots
plt.title('Harmonograph')
plt.xlabel('x(t)')
plt.ylabel('y(t)')
```

```
# saving figure
plt.savefig('output/harmonograph2.png')
```

Explanation

- In line 12, we are simulating the motion of two pendulums swinging along the x direction. Each of these pendulums has a different set of arguments, resulting in a unique pattern of motion.
- The final result can be obtained by the superposition of the two pendulums, i.e., add the motions of the two pendulums in the same direction.
- In line 13, we are doing the same thing for two pendulums' motion along the y direction.
- In lines 20 and 21, we have set the limits of the axis depending on the range of values in `x_t` and `y_t`. The axes will adjust themselves even if you change the arguments of `pendulum()` in lines 12 and 13.

Solution 3

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

# defining function for pendulum
def pendulum(t, A=1, f=1, p=0, d=0.05):
  return (A * np.sin (2 * np.pi * f * t + p) * np.exp(-1 * d * t))

# creating array for time
t = np.linspace(0, 6 * np.pi, 40000)

# generating random parameters for x pendulum
nx = rnd.randint(1, 5)          # number of x pendulums
f_x = rnd.randint(1, 40, nx)    # frequency
A_x = rnd.randint(1, 5, nx)     # amplitude
p_x = np.pi * rnd.random(nx)    # phase
d_x = 0.1 * rnd.random(nx)      # decay factor

# generating random parameters for y pendulum
ny = rnd.randint(1, 5)           # number of y pendulums
f_y = rnd.randint(1, 40, ny)     # frequency
A_y = rnd.randint(1, 5, ny)      # amplitude
p_y = np.pi * rnd.random(ny)     # phase
d_y = 0.1 * rnd.random(ny)      # decay factor
```

```
# setting initial values for x pendulum and y pendulum
x_t = 0
y_t = 0

# generating data for x pendulum
for i in range(nx):
  x_t = x_t + pendulum(t, A_x[i], f_x[i], p_x[i], d_x[i])

# generating data for y pendulum
for i in range(ny):
  y_t = y_t + pendulum(t, A_y[i], f_y[i], p_y[i], d_y[i])

plt.figure(figsize=(12, 9), dpi=100)
slc = rnd.randint(1, 4)     # number of slicings

# creating plots according to the number of slicings
for i in range(slc):
  col = rnd.random(3)
  lower = int(len(t) / slc * i)           # lower limit for index slicing
  upper = int(len(t) / slc * (i + 1))     # upper limit for index slicing

  plt.plot(x_t[lower:upper], y_t[lower:upper], color=col, linewidth='0.25')

# Adding labels
plt.title('Harmonograph')
plt.xlabel('x')
plt.ylabel('y')

# setting limits for plots
plt.xlim(min(x_t)-0.5 , max(x_t)+0.5)
plt.ylim(min(y_t)-0.5,  max(y_t)+0.5)

# saving figure
plt.savefig('output/harmonograph3.png')
```

Explanation

- On line 13, we are generating a random number of pendulums along the x direction. This number can be in the range $1 \leq n < 5$.
- In lines 14 - 17, we are generating a random set of arguments for pendulums' motion along the x direction.
- In lines 20 - 24, we are doing the same for pendulums' motion along the y direction.
- In line 27 - 28, we set the initial values of `x_t` and `y_t` equal to `0`.
- In lines 31 - 32, we are simulating the pendulums' motion along the x

direction depending on how many pendulums are generated in line 13.

- In lines 35 - 36, we have the same thing happening for the pendulums along the y direction.
- In line 39, we are generating a random number for slicing `x_t` and `y_t` so we can plot each slice with a different color.
- In the `for` loop (lines 42 - 47), we generate an array of random numbers for the `color` argument of the `plot` command.
- In lines 44 - 45, we set the lower and upper limits of the slice depending on the value of `t` and `slc`.
- In line 47, we plot each slice with a randomly generated color `col`.

This is it for the applications part of the course. In the next lesson, we will conclude this course.