PHYS2600: Homework 3*

Due on Tuesday, Feb, 27th 2018

Yangrui Hu

*This Latex template is from http://www.latextemplates.com

Contents

Problem 1

Problem: Write a program to calculate and compare the behavior of two, nearly identical, nonlinear pendulums. Use it to calculate the divergence of two nearby trajectories ($\Delta\theta$) in the chaotic regime and make a qualitative estimate of the corresponding Lyapunov exponent from the slope of a plot of $\log(\Delta\theta)$ as a function of t. Reasonable parameters to place the pendulum in the chaotic regime are: $\mathrm{l} = \mathrm{g} = 9.8$, $\Omega_D = 0.6$, $F_D = 1.1$, $\nu = 0.5$.

Solution: In this question, I write a program as Listing 1 to simulate the motion of two identical nonlinear pendulums. These two pendulums are subject to the same forces while there is a little differences between their initial conditions. I set the initial conditions for pendulum 1 is: $\theta(t = 0) = 0.6,\ \dot{\theta}(t = 0) = 0.0$ and for pendulum 2 is: $\theta(t = 0) = 0.61,\ \dot{\theta}(t = 0) = 0.0$. Although the difference between their initial angles is tiny, $\Delta\theta(t)$ increases exponentially over time. Similarly, if I change their initial angular velocities: $\theta_1(t = 0) = 0.6,\ \dot{\theta}_1(t = 0) = 0.0$; $\theta_2(t = 0) = 0.6,\ \dot{\theta}_2(t = 0) = 0.01$, $\Delta\omega(t)$ will also increases exponentially. Figure 1 shows the difference in θ and Figure 2 shows the difference in ω.

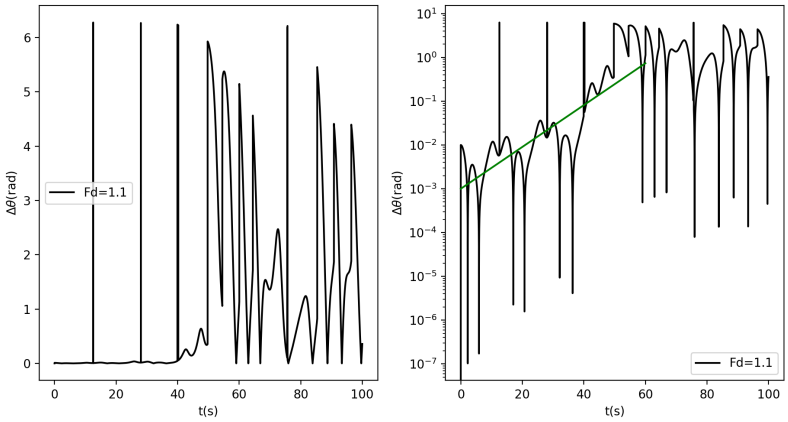

Figure 1: The divergence of two nearby trajectories ($\Delta\theta$) in the chaotic regime($F_d = 1.1$)

According to the trend lines in Figure 1 and Figure 2, the slope of the trend lines are both 0.11. So a qualitative estimate of the corresponding Lyapunov exponent is $\lambda \sim 0.11 s^{-1}$

Problem 1 continued on next page...

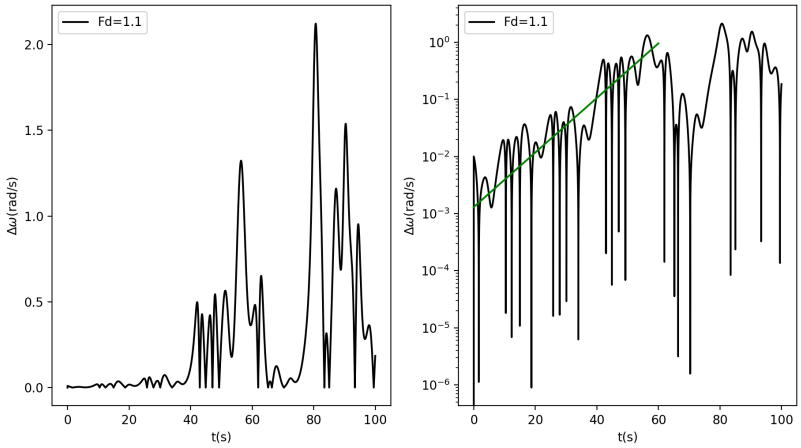

Figure 2: The divergence of two nearby trajectories ($\Delta\omega$) in the chaotic regime($F_d = 1.1$)

Listing 1: Original Script of Problem 1

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as np

class TwoPendulums (object):

    def __init__(self, l=9.8, nu=0.5, Fd=1.1, omega_d = 0.6, m = 1.0, x10 = 0.6,
     x20 = 0.61, v10 = 0.0, v20=0.0, tf = 100.0, dt = 0.001):

        g = 9.8
        omega0 = np.sqrt(g/l)
        self.l = l # length
        self.m = m # mass
        self.Fd = Fd # driving force, in units of mg
        self.omega_d = omega_d #driving frequency, in units of omega0
        self.nu = nu # viscous damping
        self.omega0 = omega0 # natural frequency

        self.x1 = x10
        self.x2 = x20
        self.v1 = v10
        self.v2 = v20

        self.tf = tf
        self.dt = dt

        npoints = int(tf/dt) # always starting at t = 0.0
        self.npoints = npoints
        self.tarray = np.linspace(0.0, tf,npoints, endpoint = True)
        npoints = int(tf/dt) # always starting at t = 0.0
        self.xv10 = np.array([self.x1, self.v1])
        self.xv20 = np.array([self.x2, self.v2])
```

Problem 1 continued on next page...

```
    def F(self, x, v, t):
        g = 9.8
        F = self.Fd*np.cos(self.omega_d*t) - self.nu*v - g/self.l*np.sin(x)
        return F

    def derivative(self, xv, t):
        x =xv[0]
        v =xv[1]
        a = self.F(x, v, t) / self.m
        return np.ravel(np.array([v, a]))

    def Differences(self):

        xv1 = odeint(self.derivative, self.xv10, self.tarray)
        x1 = xv1[:,0]
        v1 = xv1[:,1]
        x1_new = np.zeros(np.shape(x1))
        x1_new[0] = x1[0]

        dx1 = np.diff(x1)
        nx1 = np.shape(x1)[0]

        for ii in range(1,nx1):
            x1_new[ii] = x1_new[ii-1]+dx1[ii-1]
            if x1_new[ii] > np.pi:
                x1_new[ii] -= 2*np.pi
            elif x1_new[ii] < -np.pi:
                x1_new[ii] += 2*np.pi

        xv2 = odeint(self.derivative, self.xv20, self.tarray)
        x2 = xv2[:,0]
        v2 = xv2[:,1]
        x2_new = np.zeros(np.shape(x2))
        x2_new[0] = x2[0]

        dx2 = np.diff(x2)
        nx2 = np.shape(x2)[0]

        for ii in range(1,nx2):

            x2_new[ii] = x2_new[ii-1]+dx2[ii-1]
            if x2_new[ii] > np.pi:
                x2_new[ii] -= 2*np.pi
            elif x2_new[ii] < -np.pi:
                x2_new[ii] += 2*np.pi

        xv1_unwrap = xv1
        xv1[:,0] = x1_new
        xv2_unwrap = xv2
        xv2[:,0] = x2_new

        delta_x = np.zeros(min(nx1,nx2))
```

Problem 1 continued on next page...

```
            for ii in range(1,min(nx1,nx2)):
                    delta_x[ii] = abs(xv2[ii,0]-xv1[ii,0])
            testx = np.linspace(0.0, 60, 6000, endpoint = True)
            fig = plt.figure()
            ax1 = fig.add_subplot(121)
            ax2 = fig.add_subplot(122)
            ax1.plot(self.tarray,delta_x,"k",label = 'Fd=1.1')
            ax2.plot(self.tarray,delta_x,"k",label = 'Fd=1.1')
            ax2.plot(testx,0.001*np.exp(testx*0.11),"green")
            ax2.set_yscale('log')
            ax1.set_xlabel('t(s)')
            ax1.set_ylabel('$\Delta\\theta$(rad)')
            ax2.set_xlabel('t(s)')
            ax2.set_ylabel('$\Delta\\theta$(rad)')
            ax1.legend()
            ax2.legend()
            plt.show()


p = TwoPendulums()
p.Differences()
```

Problem 2

Problem: Calculate Poincare sections for the pendulum as it undergoes the period doubling route to chaos. Plot ω versus θ, with one point plotted for each drive cycle. Do this for $F_D = 1.4,\ 1.44,\ 1.465,\ 1.481,$ and 1.49 using the parameters $l = g = 9.8$, $\Omega_D = 2/3$, $\nu = 0.5$, $\omega(0) = 0$ and $\theta(0) = 0.2$. You should find that the attractor in the period 1 regime will contain only a single point. Likewise, if the behavior is period n the attractor will contain n discrete points. Be careful to remove the points corresponding to the initial transient. You must also be careful to account for the fact that time increases in discrete steps of Δt, so you will actually extract points for your Poincare section when $|t2n\pi/\Omega_D| < \Delta t/2$. Use a step size of $\Delta t = 0.005$ for this problem.

Solution: Listing 2 is my code for this question. Figure 3, Figure 4, Figure 5, Figure 6, and Figure 7 are trajectories in the phase space and Poincare sections for the pendulum with $F_D = 1.4,\ 1.44,\ 1.465,\ 1.481,$ and 1.49 respectively. From these figures, we can find that

- When $F_D = 1.4$, there is one point in Poincare section which means the behavior is period 1.
- When $F_D = 1.44$, there are two points and the behavior is period 2.
- When $F_D = 1.465$, there are four points and the behavior is period 4.
- When $F_D = 1.481$, there are infinity points and the behavior is chaos.
- When $F_D = 1.49$, there are two points and the behavior is back to period 2.

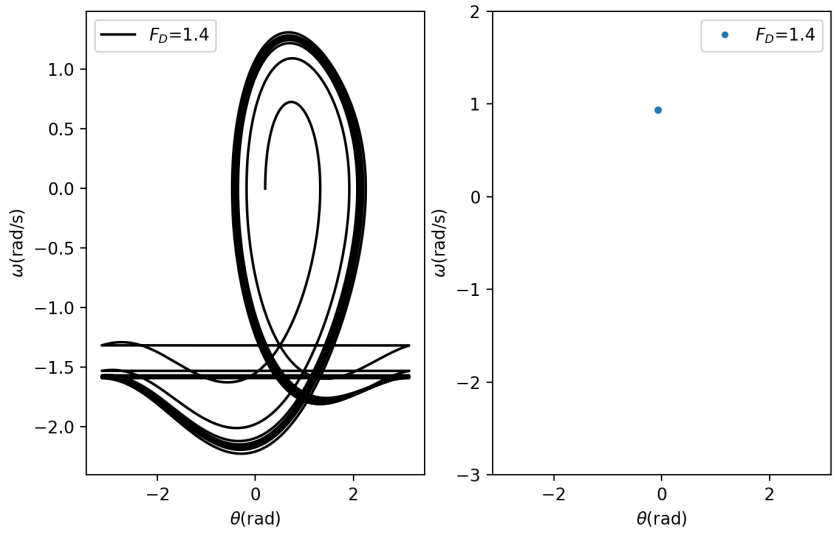

Figure 3: Trajectory in phase space (left) and Poincare section (right) of the pendulum for $F_D = 1.4$

Problem 2 continued on next page...

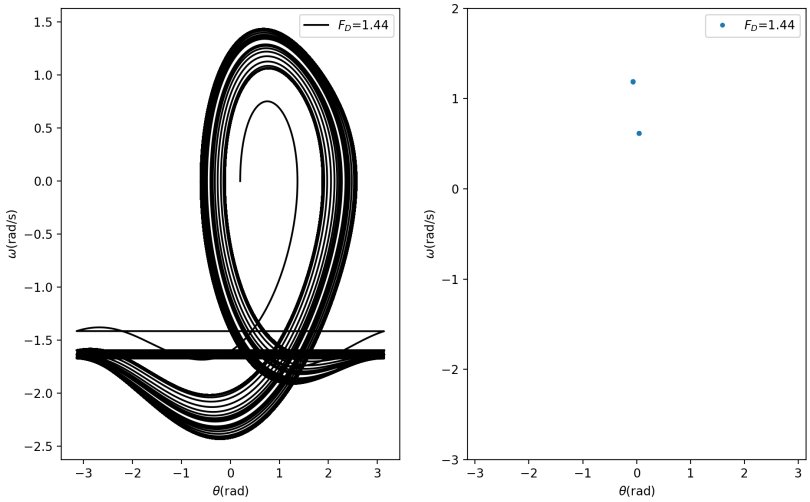

Figure 4: Trajectory in phase space (left) and Poincare section (right) of the pendulum for $F_D = 1.44$

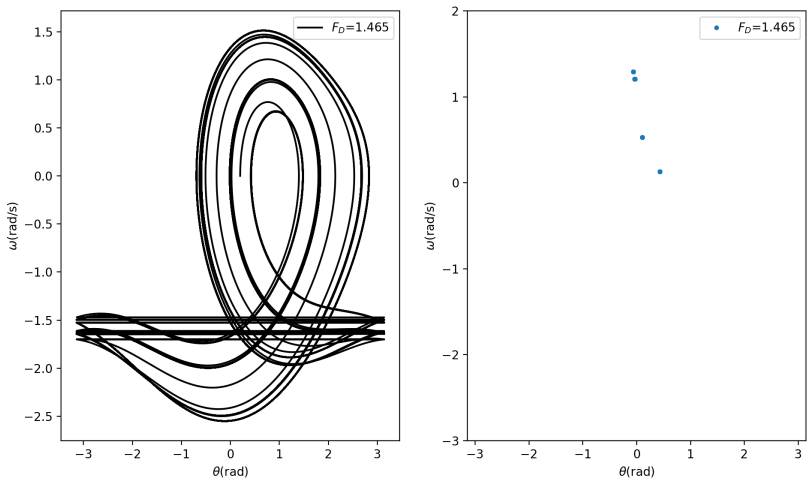

Figure 5: Trajectory in phase space (left) and Poincare section (right) of the pendulum for $F_D = 1.465$

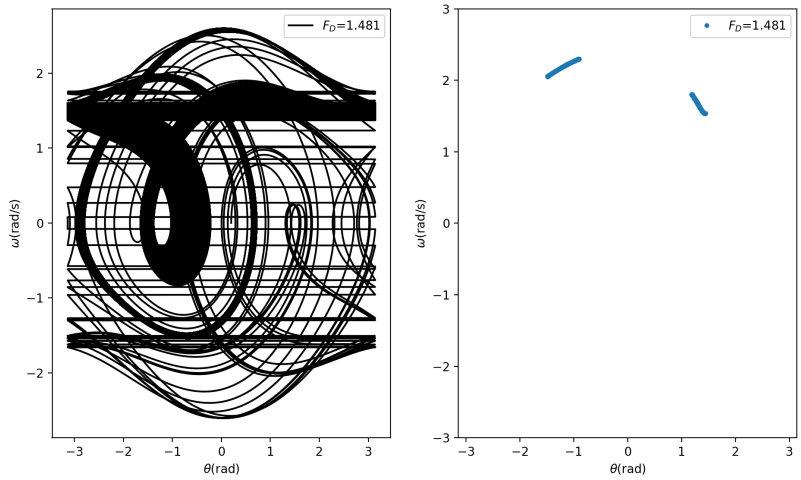

Figure 6: Trajectory in phase space (left) and Poincare section (right) of the pendulum for $F_D = 1.481$

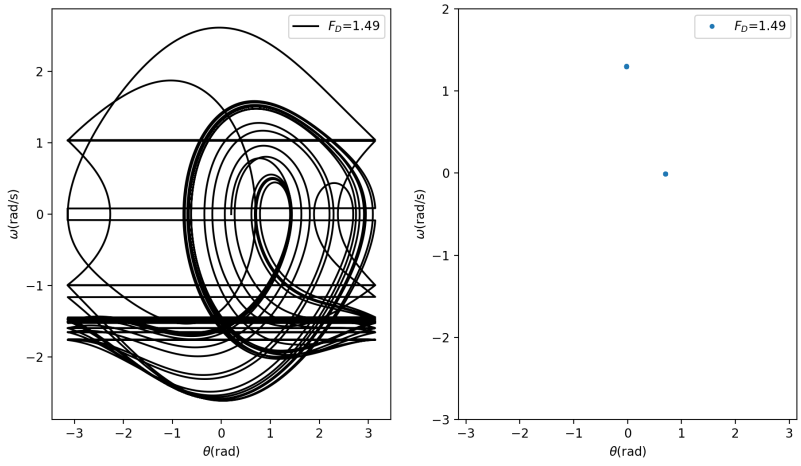

Figure 7: Trajectory in phase space (left) and Poincare section (right) of the pendulum for $F_D = 1.49$

Listing 2: Original Script of Problem 2

```
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as np

class Pendulum(object):

    def __init__(self, l = 9.8, nu = 0.5, Fd  = 1.4, omega_d = 2/3,
     m = 1.0, x0 = 0.2 ,v0 = 0.0, tf = 10000.0, dt = 0.005):
        self.g = 9.8
        self.m = m
        self.x = x0
        self.v = v0
        self.l = l
        self.Fd = Fd
        self.omega_d = omega_d
        self.nu = nu
        self.omega0 = np.sqrt(self.g/l)

        self.t = 0.0
        self.tf = tf
        self.dt = dt

        npoints = int(tf/dt)
        self.tarray = np.linspace(0.0, tf, npoints, endpoint = True)
        self.xv0 = np.array([self.x, self.v])

    def F(self, x, v, t):
        return self.Fd*np.cos(self.omega_d*t) - self.nu*v - self.g/self.l*np.sin(x)

    def scipy_trajectory(self):
        self.xv = odeint(self.derivative, self.xv0, self.tarray)
        x = self.xv[:,0]
        x_new = np.zeros(np.shape(x))
        x_new[0] = x[0]
        dx = np.diff(x)
        nx = np.shape(x)[0]
        for ii in range(1,nx):
            x_new[ii] = x_new[ii-1]+dx[ii-1]
            if x_new[ii] > np.pi:
                x_new[ii] -= 2*np.pi
            elif x_new[ii] < -np.pi:
                x_new[ii] += 2*np.pi
        self.xv_unwrap = self.xv
        self.xv[:,0] = x_new

    def derivative(self, xv, t):
        x =xv[0]
        v =xv[1]
        a = self.F(x, v, t) / self.m
        return np.ravel(np.array([v, a]))

    def Poincare(self, transient = 30, nperiods = 100):
```

```
            T = 2*np.pi/self.omega_d

            n_transient = int(T*transient/self.dt)
            t_new = self.tarray[n_transient:]
            xv_poincare = self.xv[n_transient:,:]
            index = np.nonzero((t_new%T)<(self.dt/2.))[0]
            xv_poincare = xv_poincare[index,:]

            fig = plt.figure()
            ax1 = fig.add_subplot(121)
            ax2 = fig.add_subplot(122)
            ax1.plot(self.xv[:, 0], self.xv[:, 1], 'k',label='$F_D$=1.481')
            ax2.plot(xv_poincare[:,0],xv_poincare[:,1],'.',label='$F_D$=1.481')
            ax1.set_xlabel('$\\theta$(rad)')
            ax1.set_ylabel('$\omega$(rad/s)')
            ax2.set_xlabel('$\\theta$(rad)')
            ax2.set_ylabel('$\omega$(rad/s)')
            ax2.set_xlim(-np.pi,np.pi)
            ax2.set_ylim(-3,3)
            ax1.legend()
            ax2.legend()
            plt.show()


p=Pendulum(Fd = 1.481)
p.scipy_trajectory()
p.Poincare()
```

Problem 3

Problem: The logistic map undergoes successive period doubling as μ is increased from below 3 and finally develops fully chaotic behavior at $\mu \approx 3.56$. Compute the bifurcation diagram for the logistic map. Use numerical results for the logistic map to estimate the value of the Feigenbaum parameter δ and compare it with the value for the pendulum $\delta \approx 4.669$. You should be able to get a fairly good estimate if you look at the first eight bifurcations.

Solution: Listing 3 show my code for this question. Figure 8 is the bifurcation diagram for the logistic map. By continuously zooming in, I find the bifurcation points where its period doubles as Table 1 showing. The Feigenbaum parameter is $\delta_n = (\mu_{n-1} - \mu_{n-2})/(\mu_n - \mu_{n-1})$ and the estimated value is 4.7143. Comparing with 4.669, the error is 1%.

Table 1: Table of the bifurcation points and Feigenbaum parameter δ

n	Period	μ_n	δ_n
0	1	2.95	–
1	2	3.44102	–
2	4	3.54137	4.89307
3	8	3.56305	4.6287
4	16	3.56807	4.3187
5	32	3.56906	5.0707
6	64	3.56927	4.7143

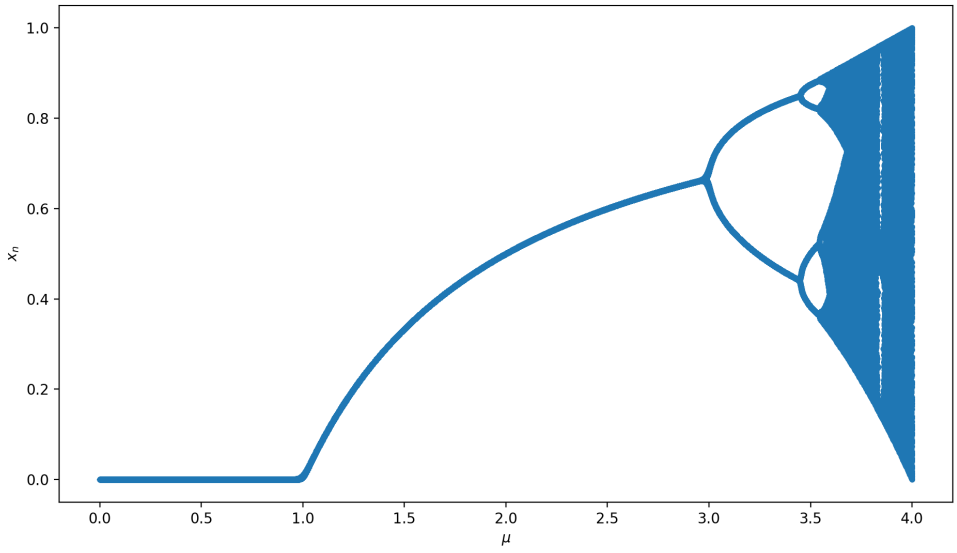

Figure 8: The bifurcation diagram of the logistic map

Problem 3 continued on next page...

Listing 3: Original Script of Problem 3

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

def logisticMap(mu,x_n):
    return mu * x_n * (1 - x_n)


def bifurcation():

    x_0 = 0.2
    N = 200
    n_attractor = 100
    mu = 0
    dmu = 0.001
    irange=list(range(1,N))
    x=[]
    y=[]
    while(mu < 4.0):
        x_n = x_0
        for i in irange:
            x_n = logisticMap(mu,x_n)
            if i > n_attractor:
                x.append(mu)
                y.append(x_n)
        mu += dmu

    # Plot the data to show the bifurcation diagram
    plt.plot(x,y,'.')
    plt.xlabel('$\mu$')
    plt.ylabel('$x_n$')
    plt.show()


bifurcation()
```

Problem 4

Problem: Study period-doubling in the Lorenz model by examining the behavior for $99.5 \leq \rho \leq 100$. Set $\sigma = 10$ and $\beta = 8/3$. Produce Poincare sections for various values of ρ in this range that show the periodicity changing.
Note that since t can only take on discrete values, there may be no time steps for which $x = 0$ exactly. To find the points where $x = 0$, you should determine when the system has crossed the plane $x = 0$ between two adjacent time steps. You can then use a linear interpolation scheme to determine the values of z and y at $x = 0$ to construct your Poincare slice. For this problem you should use the scipy odeint solver and a time step of 0.001 will be sufficient.
Bonus: Calculate and plot the bifurcation diagram for ρ in the range $90 < \rho \leq 160$.

Solution: Listing 4 is my code for this question. I set $\rho = 99.5,\ 99.7,\ 99.8,\ 99.9,\ 99.95,\ 99.98,\ 99.99,\ 100$ and calculate their 3D trajectories and Poincare sections as Figure 13 to Figure 16. From these figures, we can see that when $\rho = 99.5 - 99.8$, the system is stable and when $\rho > 99.8$ the system is chaos. Figure 17 is the bifurcation diagram of the Lorenz model. Because only when we set the range of x-axis is 1, we can see the transition from periodic to chaos, I just plot the bifurcation diagram for $99 \leq \rho \leq 100$. From the bifurcation diagram Figure 17, it's clear that the system is periodic when $\rho = 99.5 - 99.8$ and chaos when $\rho > 99.8$, which is consistent with Poincare sections.

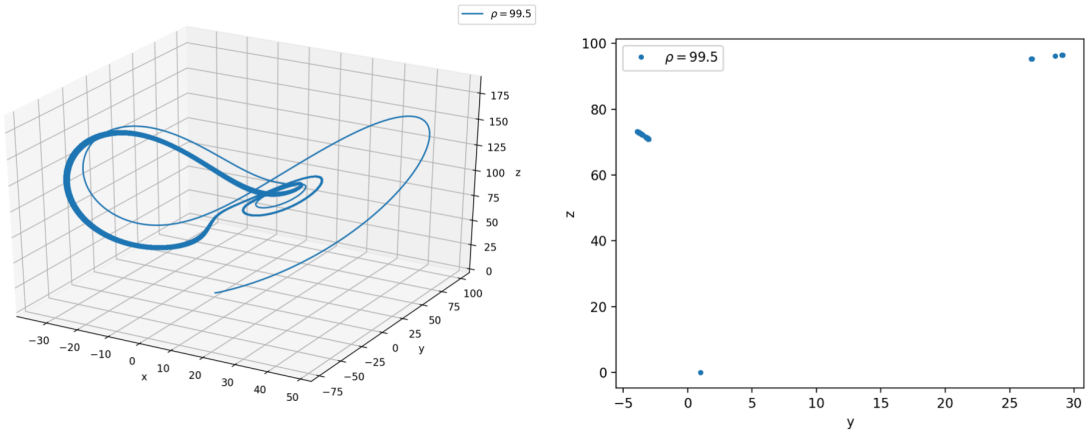

Figure 9: 3D trajectory(left) and Poincare section (right) of Lorenz model for $\rho = 99.5$

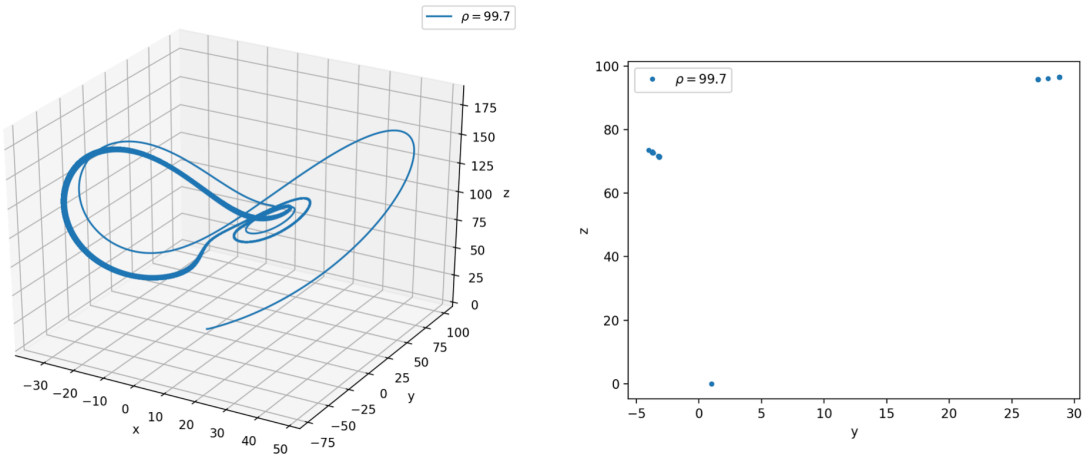

Figure 10: 3D trajectory(left) and Poincare section (right) of Lorenz model for $\rho = 99.7$

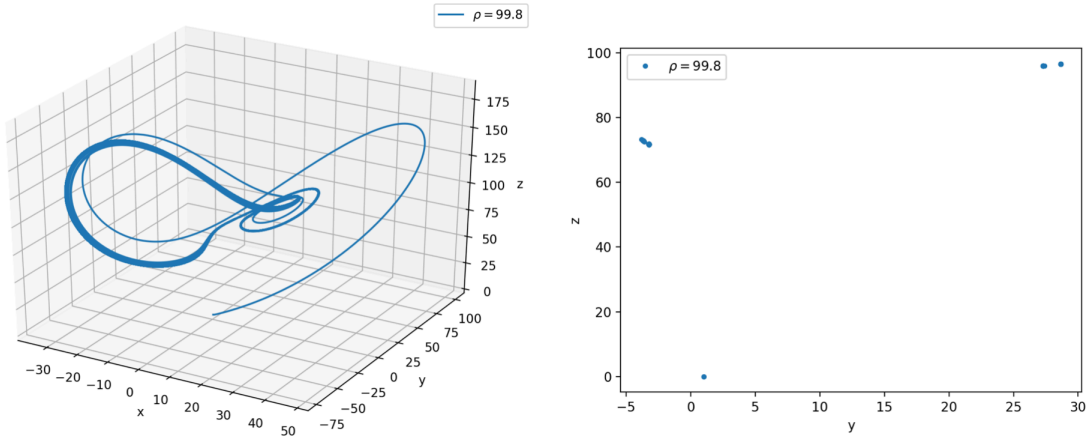

Figure 11: 3D trajectory(left) and Poincare section (right) of Lorenz model for $\rho = 99.8$

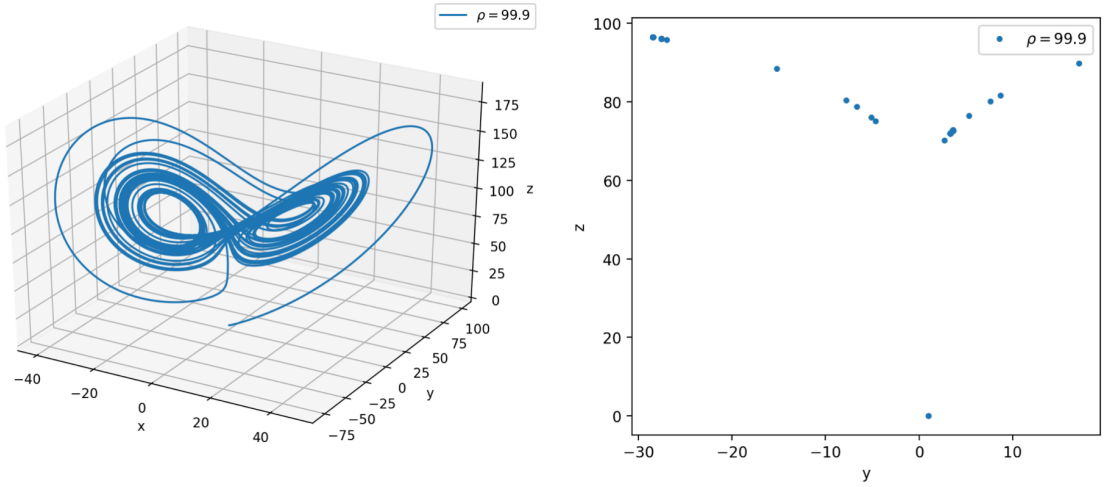

Figure 12: 3D trajectory(left) and Poincare section (right) of Lorenz model for $\rho = 99.9$

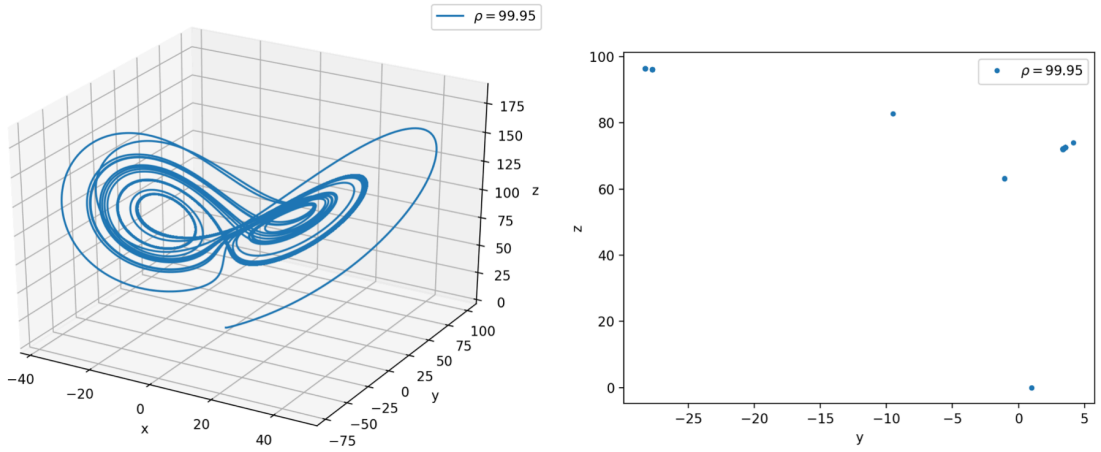

Figure 13: 3D trajectory(left) and Poincare section (right) of Lorenz model for $\rho = 99.95$

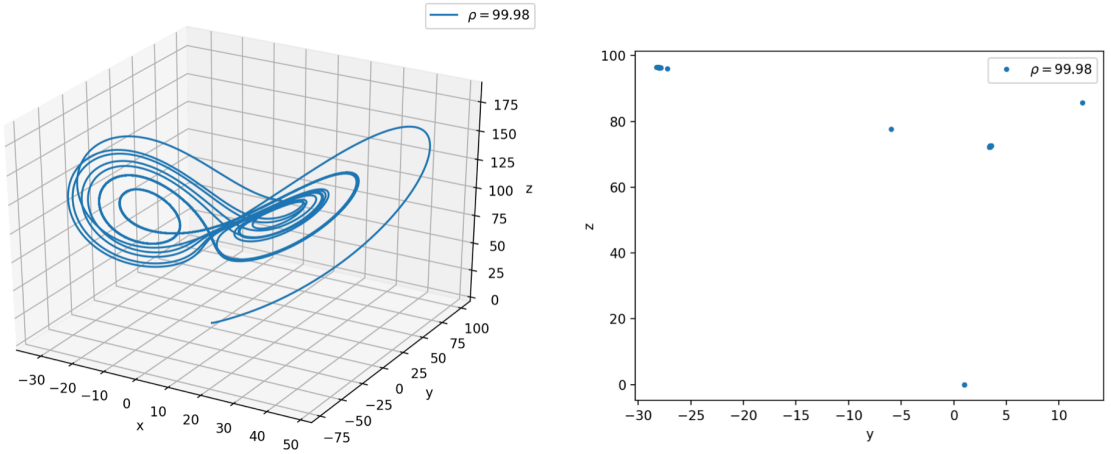

Figure 14: 3D trajectory(left) and Poincare section (right) of Lorenz model for $\rho = 99.98$

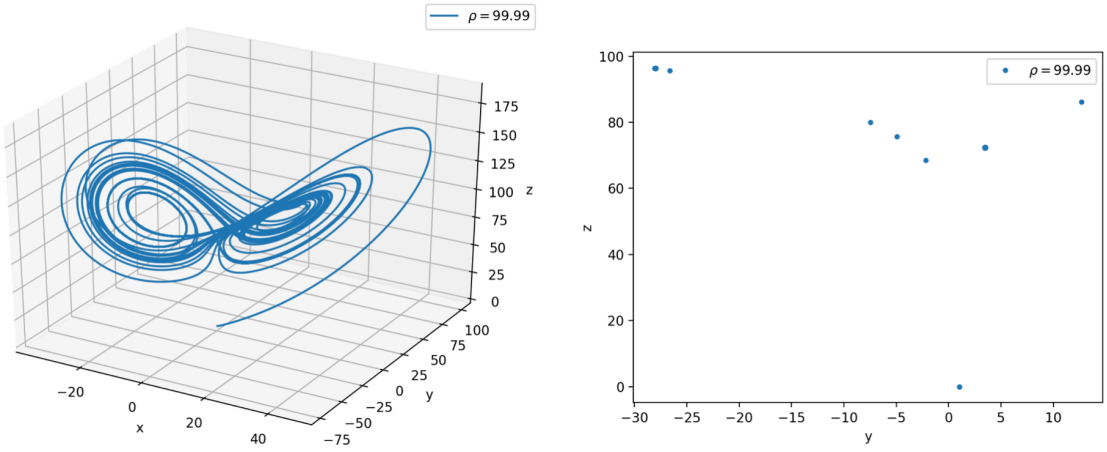

Figure 15: 3D trajectory(left) and Poincare section (right) of Lorenz model for $\rho = 99.99$

Problem 4 continued on next page. . .

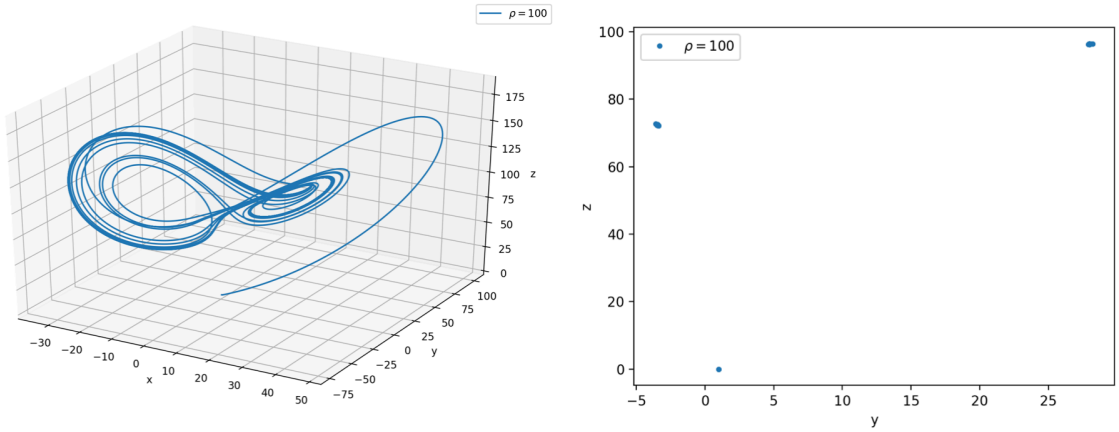

Figure 16: 3D trajectory(left) and Poincare section (right) of Lorenz model for $\rho = 100$

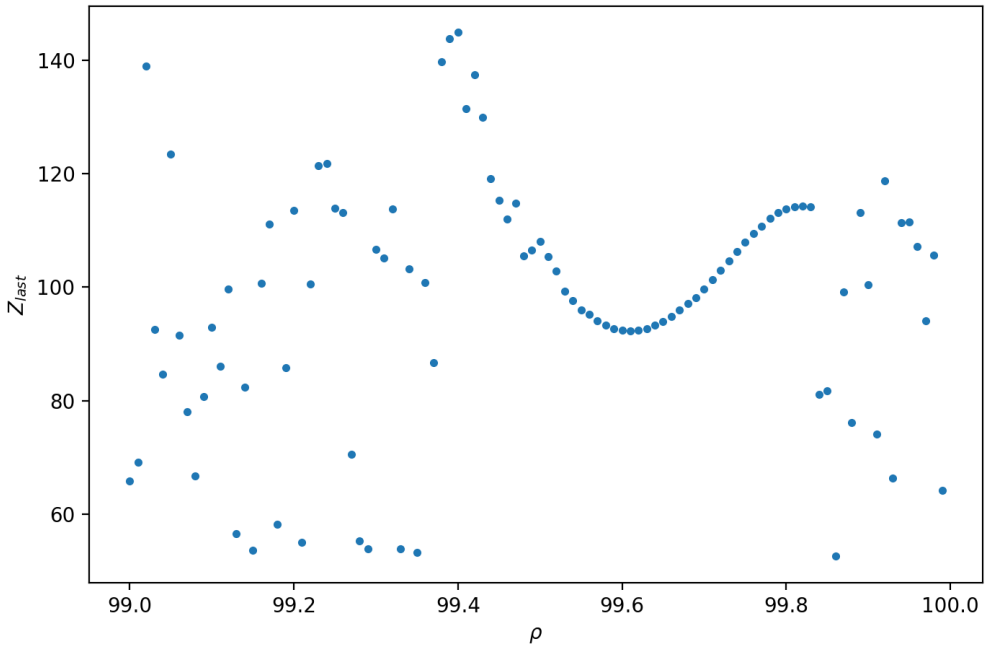

Figure 17: The bifurcation diagram of Lorenz model for $99 \leq \rho \leq 100$

Listing 4: Original script for question 4

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from scipy.integrate import odeint

class Lorenz(object):

    def __init__(self, sigma = 10, beta = 8.0/3.0, rho = 99.5, tf = 1000):
        self.t = 0.0
        self.tf = tf
        self.dt = 0.001
        self.tarray = np.linspace(0.0, self.tf, int(self.tf/self.dt),
            endpoint = True)
        self.sigma = sigma
        self.b = beta
        self.r = rho
        self.xyz0 = np.array([0,1,0]) #x=[x,y,z]

    def scipy_trajectory(self):
        self.position = odeint(self.deriv, self.xyz0, self.tarray)

    def deriv(self, xyz, t):
        x = xyz[0]
        y = xyz[1]
        z = xyz[2]
        return np.array([self.sigma*(y-x),x*(self.r-z)-y,x*y-self.b*z])

    def zlast(self):
        self.scipy_trajectory()
        a = self.position[-1,2]
        return a

    def plot3D_trajectory(self):
        fig = plt.figure()
        ax = Axes3D(fig)
        ax.plot(self.position[:,0],self.position[:,1],self.position[:,2],
            label= '$\\rho=100$')
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('z')
        ax.legend()
        plt.show()

    def Poincare(self):
        x = self.position[:,0]
        section_x = 0
        index = np.nonzero(abs(x-section_x)<0.01)[0]
        position_poincare = self.position[index,:]
        plt.plot(position_poincare[:,1],position_poincare[:,2],'.',
            label= '$\\rho=100$')

        plt.xlabel('y')
```

```
            plt.ylabel('z')
            plt.legend()
            plt.show()


def Bifurcation():
      z_last = []
      rho = []
      dr = 0.01
      r_int = 99
      r_final = 100
      r_step = int((r_final-r_int)/dr)
      for i in range(r_step):
            r = r_int + i*dr
            rho.append(r)
            P = Lorenz(tf = 100, rho = r)
            z = P.zlast()
            z_last.append(z)
      plt.plot(rho, z_last,'.')
      plt.xlabel('$\\rho$')
      plt.ylabel('$Z_{last}$')
      plt.show()


#p = Lorenz(rho = 100)
#p.scipy_trajectory()
#p.plot3D_trajectory()
#p.Poincare()

Bifurcation()
```

Problem 5

Problem: Behavior in the frequency domain. Use numpys Fast Fourier transform routines to calculate the power spectrum and examine the behavior of the nonlinear pendulum in the frequency domain. Use the following parameters $l = g = 9.8$, $\Omega_D = 2/3$, $\nu = 0.5$ for this problem.
(a) Analyze the behavior in the period-4 regime ($F_D \approx 1.465$) and show that the spectral component of $\omega(t)$ with the lowest frequency has a frequency of one-fourth the drive frequency.
(b) The bifurcation diagrams we examined in class showed that every time a period doubling threshold is crossed, a new subharmonic component is added to the $\omega(t)$ waveform. The size of this component can be extracted using a Fourier transform. Calculate $\omega(t)$ for the values of F_D between 1.4 and 1.47, near the period-2 transition. Extract the amplitude of the period-2 component as a function of F_D from the power spectra of these time series. Try to determine the functional form that describes the way in which the amplitude vanishes at the transition.

Solution: Listing 5 is my code for question 5.
(a) When $F_D = 1.465$, the trajectory and power spectrum are shown in Figure 18. Here, the peak with the lowest frequency is too tiny to distinguish, so I zoom in the figure and find the peak. According to Figure 18, the spectral component with the lowest frequency is $0.142435(rad/s)$ and its amplitude is 1.58952×10^6. The drive frequency is $\Omega_D = 2/3 \approx 0.66667$, so the one-fourth of Ω_D is 0.16667. So the spectral component with the lowest frequency approximately has a frequency of one-fourth the drive frequency.

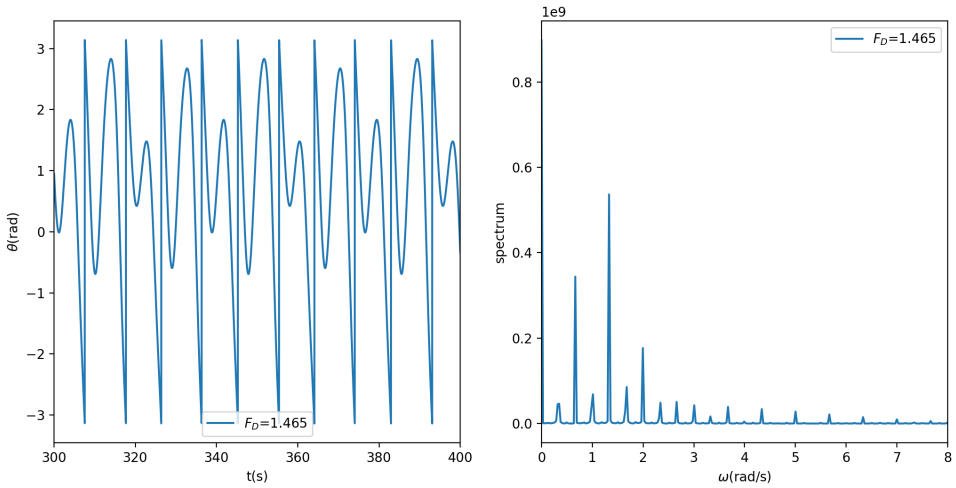

Figure 18: The trajectory and power spectrum for $F_D = 1.465$

(b) The power spectra of $F_D = 1.4,\ 1.44,\ 1.47$ are shown as Figure 19, Figure 20, and Figure 21 respectively. From these three figures, it's apparent that the period for $F_D = 1.44$ is twice as much as the period for $F_D = 1.4$ and the period for $F_D = 1.47$ is twice as much as the period for $F_D = 1.44$. And the lowest frequencies in their power spectra are approximately Ω_D, $\Omega_D/2$, $\Omega_D/4$. Extract the amplitude of the period-2 component ($\omega = \Omega_D/2$) as a function of F_D and the plot is shown as Figure 22. Its trend line is like the step function. By fitting the plot using 4-order polynomial function, the functional form that describes the way in which the amplitude vanishes at the transition is

$$\text{Amplitude}/10^{13} = -1.902 \times F_D^4 + 6.237 \times F_D^3 - 13.375 \times F_D^2 + 12.71 \times F_D - 4.5345 \quad (1)$$

Problem 5 continued on next page...

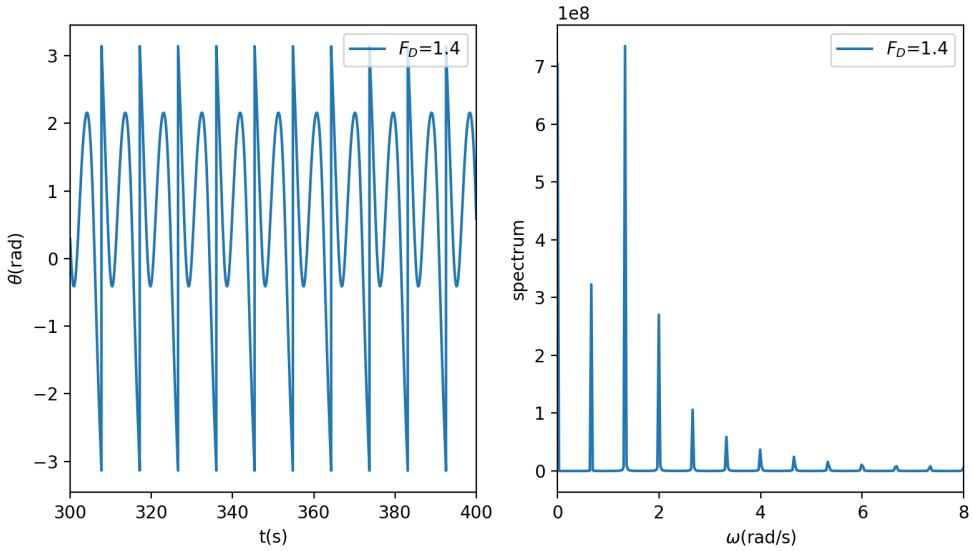

Figure 19: The trajectory and power spectrum for $F_D = 1.4$

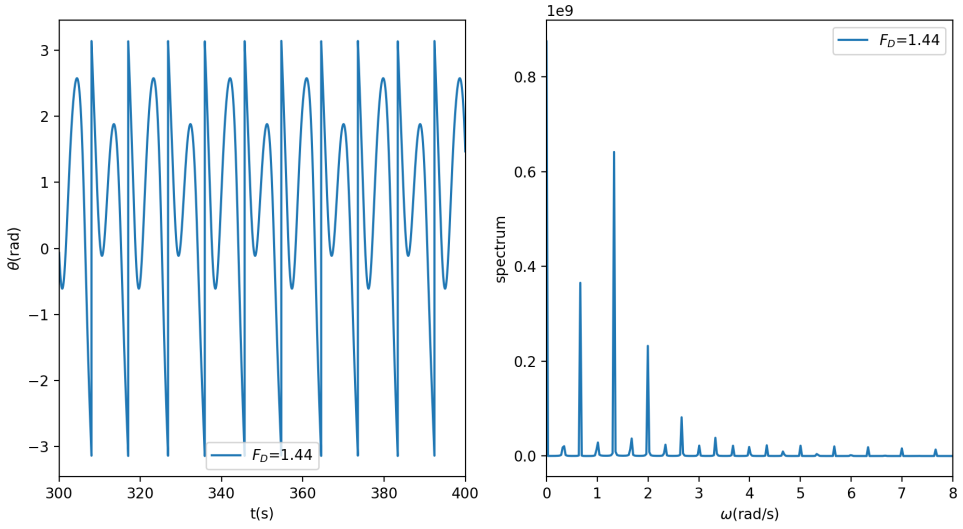

Figure 20: The trajectory and power spectrum for $F_D = 1.44$

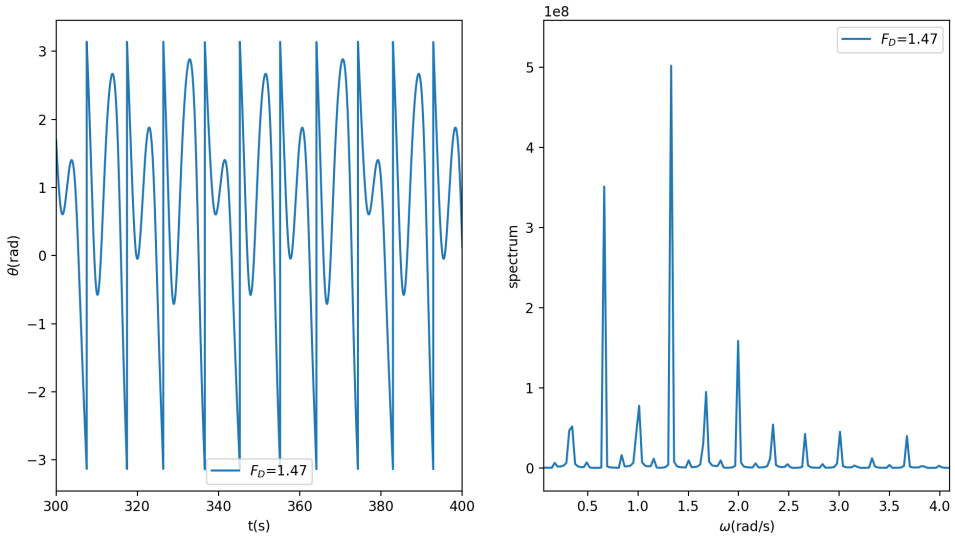

Figure 21: The trajectory and power spectrum for $F_D = 1.47$

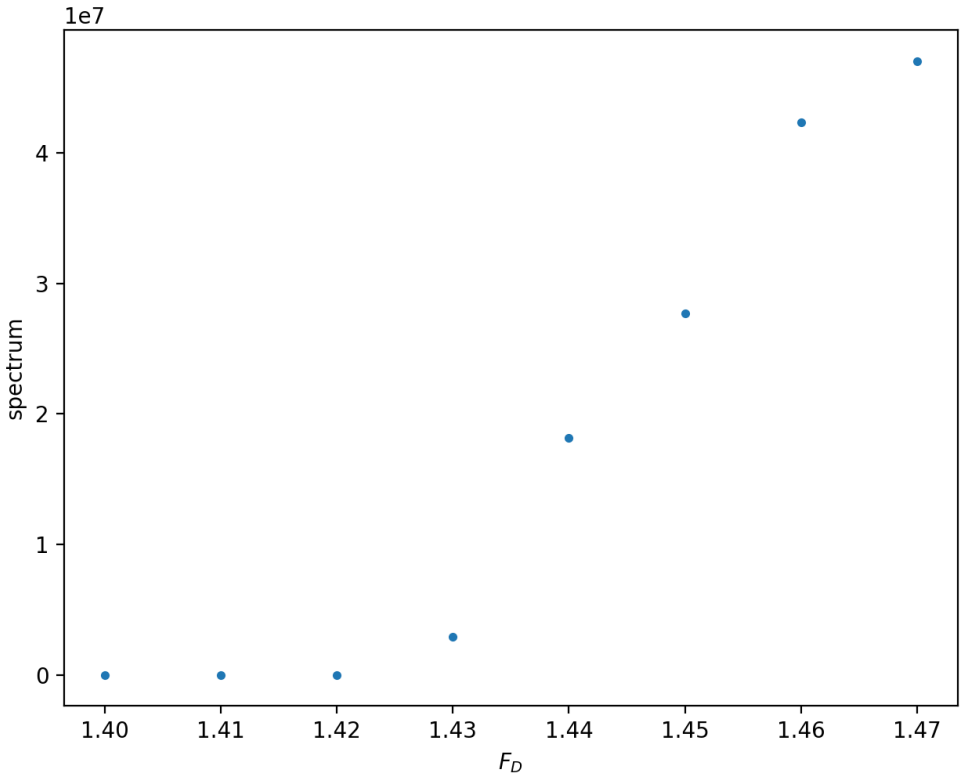

Figure 22: The plot of the amplitude of the period-2 component versus F_D

Problem 5 continued on next page...

Listing 5: Original Script of Problem 5

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

class Pendulum(object):

     def __init__(self, l = 9.8, nu = 0.5, Fd  = 1.465, omega_d = 2/3,
      m = 1.0, x0 = 0.2 ,v0 = 0.0, tf = 500.0, dt = 0.005):
          self.g = 9.8
          self.m = m
          self.x = x0
          self.v = v0
          self.l = l
          self.Fd = Fd
          self.omega_d = omega_d
          self.nu = nu
          self.omega0 = np.sqrt(self.g/l)

          self.t = 0.0
          self.tf = tf
          self.dt = dt

          npoints = int(tf/dt)
          self.tarray = np.linspace(0.0, tf, npoints, endpoint = True)
          self.xv0 = np.array([self.x, self.v])

     def F(self, x, v, t):
          return self.Fd*np.cos(self.omega_d*t) - self.nu*v
          - self.g/self.l*np.sin(x)

     def scipy_trajectory(self):
          self.xv = odeint(self.derivative, self.xv0, self.tarray)
          x = self.xv[:,0]
          x_new = np.zeros(np.shape(x))
          x_new[0] = x[0]
          dx = np.diff(x)
          nx = np.shape(x)[0]
          for ii in range(1,nx):
               x_new[ii] = x_new[ii-1]+dx[ii-1]
               if x_new[ii] > np.pi:
                    x_new[ii] -= 2*np.pi
               elif x_new[ii] < -np.pi:
                    x_new[ii] += 2*np.pi
          self.xv_unwrap = self.xv
          self.xv[:,0] = x_new

     def derivative(self, xv, t):
          x =xv[0]
          v =xv[1]
          a = self.F(x, v, t) / self.m
          return np.ravel(np.array([v, a]))
```

Problem 5 continued on next page...

```
     def powerspectrum(self):
          transient = 30
          T = 2*np.pi/self.omega_d
          n_transient = int(T*transient/self.dt)
          t_new = self.tarray[n_transient:]
          xv_new = self.xv[n_transient:,:]
          data = xv_new[:,0]
          ps = (np.abs(np.fft.fft(data)))**2
          time_step = self.dt
          freqs = 2*np.pi*np.fft.fftfreq(data.size, time_step)
          idx = np.argsort(freqs)

          fig = plt.figure()
          ax1 = fig.add_subplot(121)
          ax2 = fig.add_subplot(122)
          ax1.plot(t_new, xv_new[:,0],label='$F_D$=1.467')
          ax2.plot(freqs[idx],ps[idx],label='$F_D$=1.467')
          ax1.set_xlabel('t(s)')
          ax1.set_ylabel('$\\theta$(rad)')
          ax2.set_xlabel('$\omega$(rad/s)')
          ax2.set_ylabel('spectrum')
          ax1.set_xlim(300,400)
          ax2.set_xlim(0,8)
          ax1.legend()
          ax2.legend()
          plt.show()

def amplitude_plot():
     a = np.loadtxt('problem5-plot.txt')
     plt.plot(a[:,0],a[:,1],'.')
     plt.xlabel('$F_D$')
     plt.ylabel('spectrum')
     plt.show()

#p = Pendulum(Fd = 1.467)
#p.scipy_trajectory()
#p.powerspectrum()

amplitude_plot()
```