

# Physics 2130: Assignment III

Jeremy Favro

December 2, 2024

**Problem 1.** Consider the driven, damped harmonic oscillator. Its equation of motion is:

$$\ddot{x} + 2\beta\dot{x} + \omega_0^2 = A \cos(\omega t)$$

In the case of an underdamped oscillator, i.e.,  $\beta < \omega_0^2$  we found that the solution for the equation of motion is:

$$x(t) = x_c(t) + x_p(t)$$

Where:

$$x_c(t)e^{-\beta t} [c_1 e^{i\omega_1 t} + c_2 e^{-i\omega_1 t}] = B e^{-\beta t} \cos(\omega_1 t - \phi) = \Gamma(t)s(t)$$

And where:

$$\omega_1 = \sqrt{\omega_0^2 - \beta^2}$$

$$\Gamma(t) = B e^{-\beta t}$$

$$s(t) = \cos(\omega_1 t - \phi)$$

The particular solution is instead:

$$x_p(t) = D \cos(\omega t - \delta)$$

Where:

$$D = \frac{A}{\sqrt{(\omega_0^2 - \omega^2)^2 + 4\omega^2\beta^2}}$$

$$\delta = \arctan\left(\frac{2\omega\beta}{\omega_0^2 - \omega^2}\right)$$

- Consider an underdamped driven oscillator that starts with the initial conditions  $x(t=0) = x_0$  and  $\dot{x}(t=0) = 0$ . Find the analytical expressions for the unknown coefficients in  $x(t)$  using these initial conditions.
- Write a python program that returns (and prints) the values of  $\omega_1$ ,  $\beta$ ,  $\phi$ ,  $D$  and  $\delta$  for a given  $x_0$ . This should be coded as a function called `harm_osc_params` that accepts as inputs  $\omega_0$ ,  $\beta$ ,  $A$ ,  $\omega$  and  $x_0$ .
- In the same python program, now write a new function, `harm_osc_x_pos`, that calculates the array of positions  $x$  of the harmonic oscillator for a given array of times  $t$ . This function should receive the array  $t$  as an input as well as the values of  $\omega_0$ ,  $\beta$ ,  $A$ ,  $\omega$  and  $x_0$ . It should call the previously written function `harm_osc_params` for the calculations of all the oscillation parameters. It should return the array of positions  $x$  of the harmonic oscillator for each value of  $t$ .
- In the same python program, now write a new function to plot the data, `harm_osc_x_plot_single`. The function receives as inputs the arrays  $t$  and  $x$  generated at the previous step.
- Pick three values of  $\beta$  (remember of the constraint  $\beta < \omega_0$ ) and plot in the same graph  $x(t)$  for the three chosen values. For reference use  $\omega_0 = 1\text{s}^{-1}$  and  $A = 1\text{s}^{-2}$  (but play with the values of  $A$  to see the relative importance of the driver and the damping). Comment on what effect  $\beta$  has on both the transient and steady-state solution.
- To help you distinguish the different effects, now write a function called `harm_osc_damp_drive` that plots in the same graph,  $s(t)$ ,  $\Gamma(t)$ ,  $x_c(t)$  and  $x_p(t)$ . Comment on the results, specifically on the contribution of each component.

- (g) Finally, write a new function called `harm_osc_euler_cromer` that receives has input parameters  $\omega_0$ ,  $\beta$ ,  $A$ ,  $\omega$ ,  $x_0$ , and the analytical solution  $x(t)$ . This function should calculate a new  $x(t)$ , called  $x_{EC}(t)$ , that uses the Euler-Cromer method to determine the position of the oscillator as a function of time for the same initial conditions. Additionally, the function should plot, on the same graph, the solutions  $x(t)$  and  $x_{EC}(t)$  obtained with the analytical and Euler-Cromer methods, respectively, and the residuals, i.e., the difference  $x(t) - x_{EC}(t)$ . Discuss how you choose a value of  $\Delta t$  that gives a sufficiently accurate answer (which means defining “sufficiently accurate”).

### Solution 1.

- (a)

$$\begin{aligned} x(t=0) &= B \cos(-\phi) + D \cos(-\delta) = x_0 \\ \Rightarrow x_0 - D \cos(\delta) &= B \cos(\phi) \\ \Rightarrow \frac{x_0 - D \cos(\delta)}{\cos(\phi)} &= B \end{aligned}$$

$$\begin{aligned} \dot{x}(t=0) &= -B\omega_1 \sin(-\phi) - B\beta \cos(-\phi) - D\omega \sin(-\delta) = 0 \\ &= B\omega_1 \sin(\phi) - B\beta \cos(\phi) + D\omega \sin(\delta) \\ &= \frac{x_0 - D \cos(\delta)}{\cos(\phi)} \omega_1 \sin(\phi) - \frac{x_0 - D \cos(\delta)}{\cos(\phi)} \beta \cos(\phi) + D\omega \sin(\delta) \\ &= (x_0 - D \cos(\delta)) \omega_1 \tan(\phi) - (x_0 - D \cos(\delta)) \beta + D\omega \sin(\delta) \\ \Rightarrow -\frac{D\omega \sin(\delta)}{(x_0 - D \cos(\delta))} &= \omega_1 \tan(\phi) - \beta \\ \Rightarrow \arctan\left(\frac{\beta - \frac{D\omega \sin(\delta)}{(x_0 - D \cos(\delta))}}{\omega_1}\right) &= \phi \end{aligned}$$

- (b) `import numpy as np`

```
def harm_osc_params(w_0: float,
                    beta: float,
                    A: float,
                    w: float,
                    x_0: float
                    ) -> tuple[float, float, float, float, float]:

    # "Given"
    w_1 = np.sqrt(w_0**2 - beta**2)
    D = A / (np.sqrt((w_0**2 - w**2)**2 + 4 * (w**2) * (beta**2)))
    delta = np.arctan2((2 * w * beta), (w_0**2 - w**2))

    # Unknowns
    phi = np.arctan2(
        ((beta - (D * w * np.sin(delta))) / (x_0 - D * np.cos(delta))),
        w_1)
    B = (x_0 - D * np.cos(delta)) / (np.cos(phi))

    print(f"w_1: {w_1}", f"B: {B}", f"phi: {phi}",
          f"D: {D}", f"delta: {delta}", sep="\n")

    return w_1, B, phi, D, delta
```

```

(c) import numpy as np
    import q1b

    def harm_osc_x_pos(w_0: float,
                      beta: float,
                      A: float,
                      w: float,
                      x_0: float,
                      t: np.ndarray
                      ) -> list:
        w_1, B, phi, D, delta = q1b.harm_osc_params(w_0,
                                                    beta,
                                                    A,
                                                    w,
                                                    x_0)

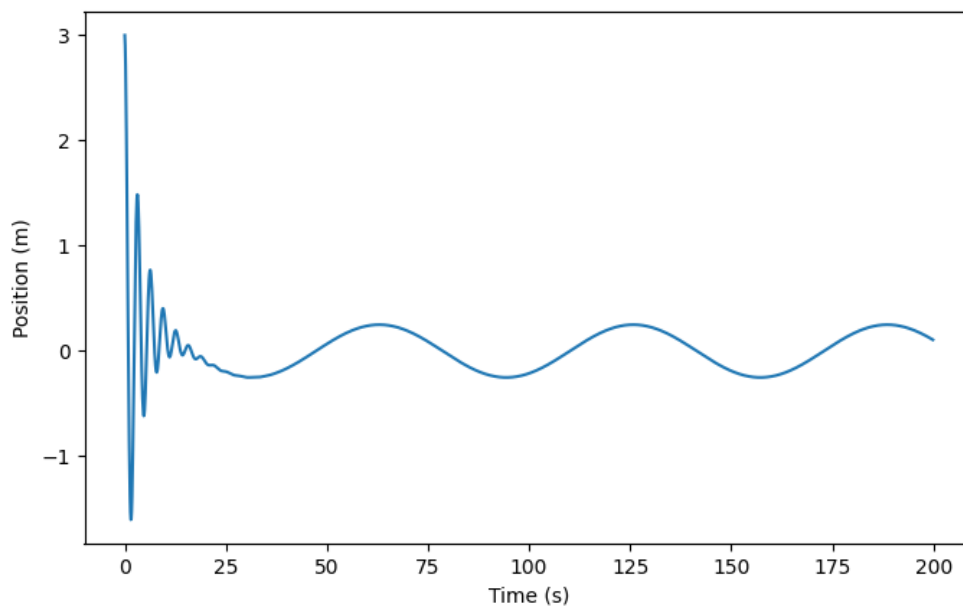
        return B*np.exp(-beta*t)*np.cos(w_1*t-phi)+D*np.cos(w*t-delta)

(d) import matplotlib.pyplot as pyplot
    import numpy as np
    import q1c

    def harm_osc_x_plot_single(x: np.ndarray, t: np.ndarray) -> None:
        pyplot.plot(t, x)
        pyplot.xlabel("Time (s)")
        pyplot.ylabel("Position (m)")
        pyplot.show()

    t_vals = np.arange(0, 200, 0.1)
    x_vals = q1c.harm_osc_x_pos(2, 0.25, 1, 0.1, 3, t_vals)
    harm_osc_x_plot_single(x_vals, t_vals)

```



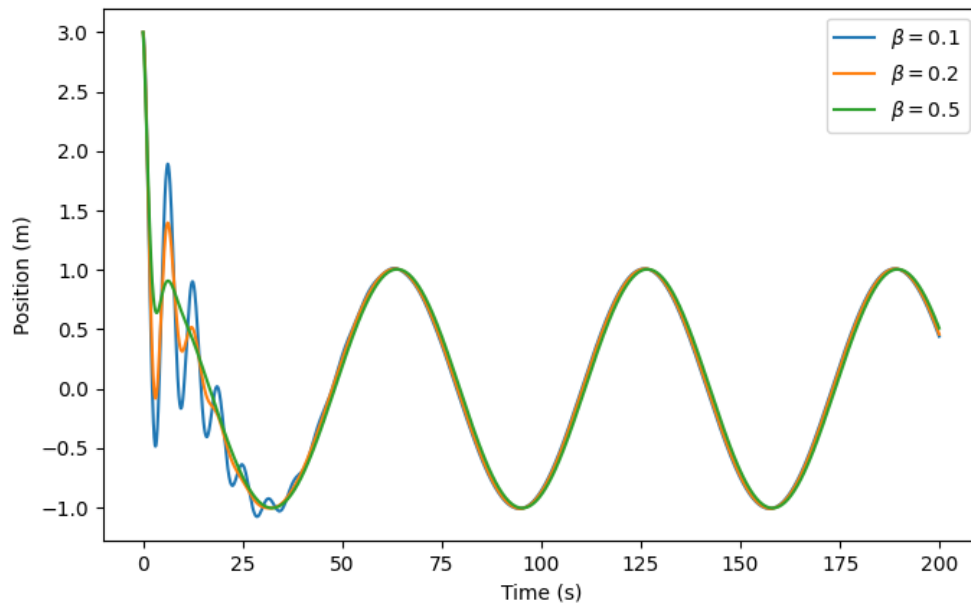
The first regime, visible as the rapid oscillations from  $t = 0$  to  $t \approx 30$  is dominated by the  $Be^{-\beta t} \cos(\omega_1 t - \phi)$  term. Because This term contains an exponential decay factor it will fade away as time progresses leaving the  $D \cos(\omega t - \delta)$  term as the only oscillator. Both the transient and steady-state regimes have, as expected, periods and therefore frequencies consistent with their respective driving equations.

```
(e) import numpy as np
import matplotlib.pyplot as pyplot
import q1c

t_vals = np.arange(0, 200, 0.1)

for beta in [0.1, 0.2, 0.5]:
    x_vals = q1c.harm_osc_x_pos(1, beta, 1, 0.1, 3, t_vals)
    pyplot.plot(t_vals, x_vals, label=rf"$\beta$={beta}")

pyplot.xlabel("Time (s)")
pyplot.ylabel("Position (m)")
pyplot.legend()
pyplot.show()
```



As is expected here, greater values of  $\beta$  result in quicker die-off of the transient solution due to the increased growth-rate of the exponential damping term.  $\beta$  only slightly effects the steady-state solution, causing slight displacement as the transient solution never actually reaches zero.

```
(f) import numpy as np
import matplotlib.pyplot as pyplot
import q1b

t_vals = np.arange(0, 200, 0.1)

def harm_osc_damp_drive():
    t = np.arange(0, 100, 0.1)
```

```

w_0 = 2
beta = 0.25
A = 1
w = 0.1
x_0 = 3
w_1, B, phi, D, delta = q1b.harm_osc_params(w_0,
                                              beta,
                                              A,
                                              w,
                                              x_0)

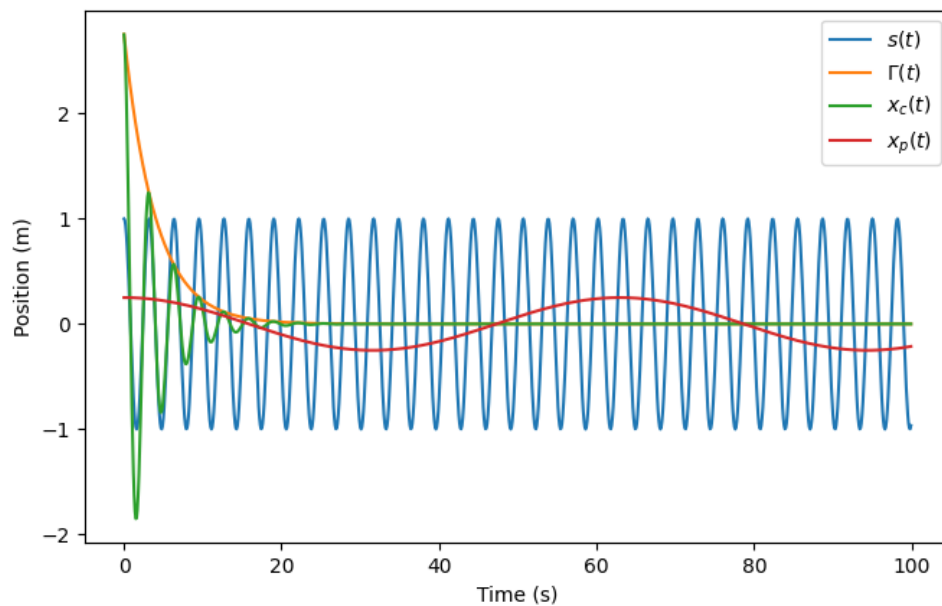
s = np.cos(w_1*t-phi)
gamma = B*np.e**(-beta*t)
x_c = s*gamma
x_p = D*np.cos(w*t-delta)

pyplot.plot(t, s, label=r"$s(t)$")
pyplot.plot(t, gamma, label=r"$\Gamma(t)$")
pyplot.plot(t, x_c, label=r"$x_c(t)$")
pyplot.plot(t, x_p, label=r"$x_p(t)$")

pyplot.xlabel("Time (s)")
pyplot.ylabel("Position (m)")
pyplot.legend()
pyplot.show()

harm_osc_damp_drive()

```



$s(t)$  contributes the oscillation of the transient solution.  $\Gamma(t)$  contributes the exponential decay envelope.  $x_c(t)$  contributes, being the product of  $s(t)$  and  $\Gamma(t)$  produces the initial decaying transient solution.  $x_p(t)$  contributes the steady-state solution.

```

(g) import numpy as np
import matplotlib.pyplot as pyplot
import q1c

def harm_osc_euler_cromer(w_0: float,
                           beta: float,
                           A: float,
                           w: float,
                           x_0: float,
                           t: np.ndarray
                           ):
    dt = t[1]-t[0]
    x = np.full_like(t, x_0)
    x_r = q1c.harm_osc_x_pos(w_0, beta, A, w, x_0, t)
    v = np.zeros_like(t)
    for t_i in range(len(t) - 1):
        a = A*np.cos(w*t[t_i])-2*beta*v[t_i]-(w_0**2)*x[t_i]

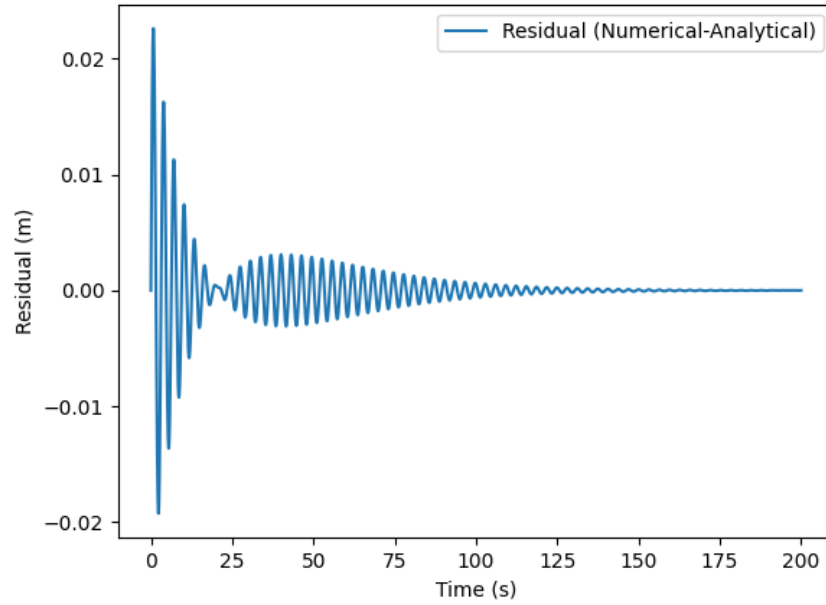
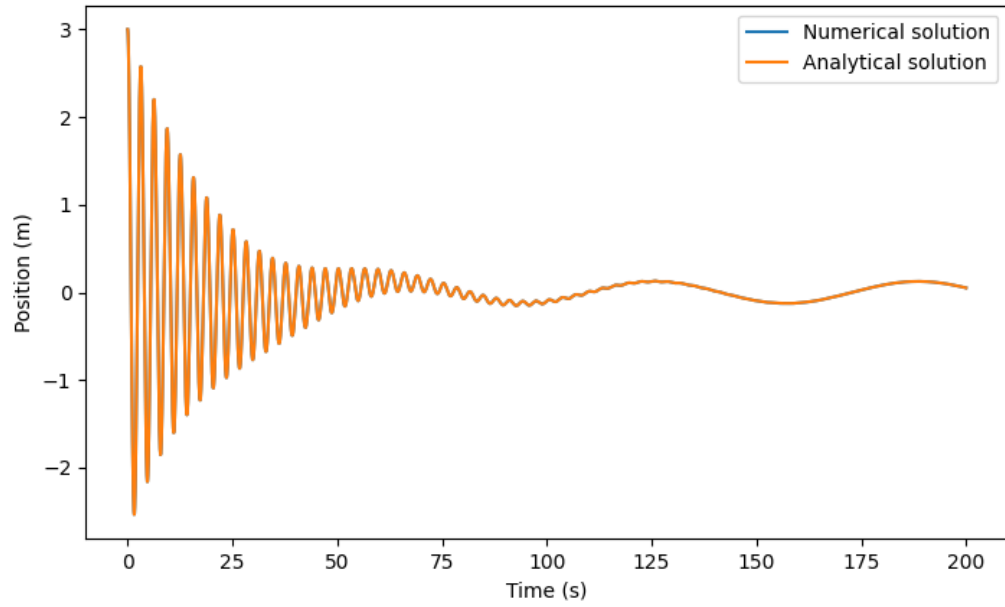
        v[t_i+1] = v[t_i] + a*dt
        x[t_i+1] = x[t_i] + v[t_i+1]*dt

    pyplot.plot(t, x, label="Numerical solution")
    pyplot.plot(t, x_r, label="Analytical solution")
    pyplot.xlabel("Time (s)")
    pyplot.ylabel("Position (m)")
    pyplot.legend()
    pyplot.figure(0)

    pyplot.plot(t, x-x_r, label="Residual (Numerical-Analytical)")
    pyplot.xlabel("Time (s)")
    pyplot.ylabel("Residual (m)")
    pyplot.legend()
    pyplot.show()

t_vals = np.arange(0, 200, 1/128)
harm_osc_euler_cromer(2, 0.05, 1/2, 0.1, 3, t_vals)

```



Note that here I plotted the residuals and numerical-analytical solutions on different graphs due to the difference in amplitude of each. I chose a  $\Delta t$  of  $\frac{1}{128}$ s as it is the point at which reciprocals of  $2^n$  (easily represented in base 2) begin to show strong (maximum residual  $\approx 0.02$ m) predictive power for both the steady state and transient conditions. Some larger values of  $\Delta t$  such as  $\frac{1}{64}$  result in a lower maximum residual but do not exhibit the same generally high predictive power seen in smaller timesteps. I specifically sought residuals  $< 0.05$  as the given conditions are only specified to at most two digits of accuracy so a result with similar accuracy is “reasonably” accurate.

**Problem 2.** We want to study now the behaviour of the **non-linear (physical) pendulum**. The analytical solutions we generally get for oscillating systems are derived under the small-angle approximation, which allows us to write  $\sin(\theta) \approx \theta$ . If one drops this hypothesis things change, and the motion now becomes dependent on the amplitude. The equation of motion is now:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l} \sin(\theta) - q \frac{d\theta}{dt} + F_d \sin(\Omega_D t)$$

Where the only difference from the linear case is that we now have  $\sin \theta$  instead of  $\theta$ . This equation of motion has no analytical solutions; therefore, we need to solve it numerically. We thus need to write the equations for angular acceleration and velocity and use them, e.g., with the Euler-Cromer method:

$$\frac{d\omega}{dt} = -\frac{g}{l} \sin(\theta) - q \frac{d\theta}{dt} + F_d \sin(\Omega_D t)$$

$$\frac{d\theta}{dt} = \omega$$

- Write a python program that calculates numerically  $\theta(t)$  using the Euler-Cromer method.
- Write another python program (you can mostly base it on what you wrote in (2a)) that calculates two separate solutions  $\theta_1(t)$  and  $\theta_2(t)$ , where  $\theta_1(t)$  starts with an initial angle  $\theta_0 = 10^\circ$  while  $\theta_2(t)$  starts with an ever so slightly different initial angle  $\theta_0 = 10.05^\circ$ .
- In the program you wrote in (2b), add a phase-space plot where you now plot  $\omega$  vs.  $\theta$ .

### Solution 2.

```
(a) import numpy as np
import matplotlib.pyplot as pyplot

g = 9.8
t = np.arange(0, 200, 0.1)

def euler_chromer_nonlinear(theta_0: float,
                             w_0: float,
                             l: float,
                             q: float,
                             F_d: float,
                             W: float,
                             t_vals: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    theta = np.full_like(t_vals, np.deg2rad(theta_0))
    omega = np.full_like(t_vals, w_0)
    dt = t_vals[1]-t_vals[0]
    for t_i in range(len(t_vals) - 1):
        omega[t_i+1] = omega[t_i] + \
            (-(g/l)*np.sin(theta[t_i])-q *
             omega[t_i] + F_d*np.sin(W*t_vals[t_i]))*dt
        theta[t_i+1] = theta[t_i]+omega[t_i+1]*dt

    return theta

pyplot.figure(0)
pyplot.title(r"$F_D=0.5$")
for i in [10, 25, 45]:
    pyplot.plot(t, euler_chromer_nonlinear(
        i, 0, g, 0.5, 0.5, 2/3, t), label=r"$\theta={i}^\circ$")
pyplot.xlabel("Time (s)")
pyplot.ylabel("Position (rad)")
```



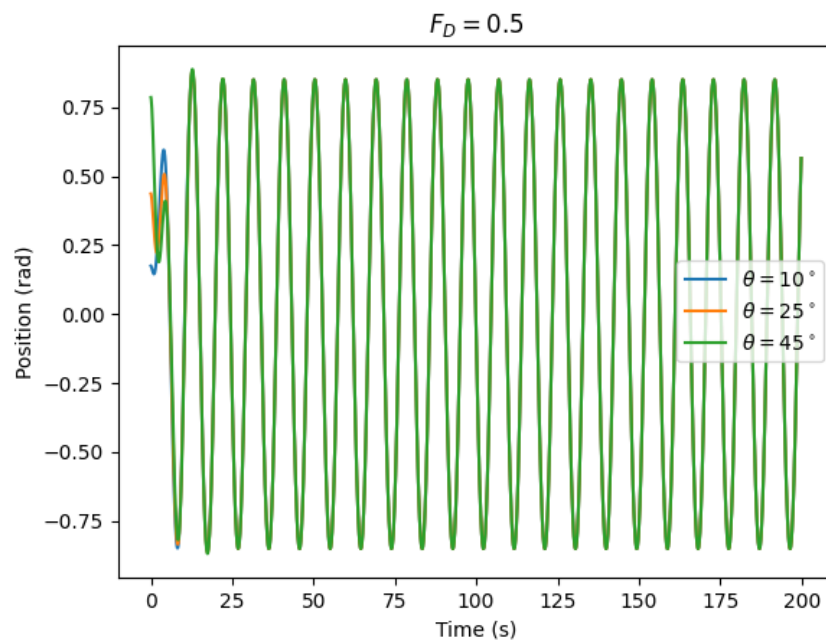
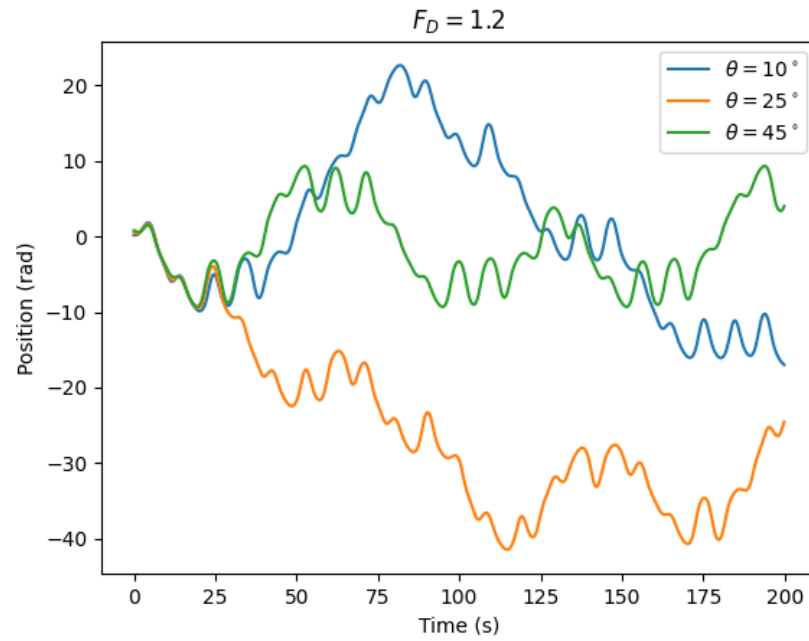
```

pyplot.legend()

pyplot.figure(1)
pyplot.title(r"$F_D=1.2$")
for i in [10, 25, 45]:
    pyplot.plot(t, euler_chromer_nonlinear(
        i, 0, g, 0.5, 1.2, 2/3, t), label=r"$\theta={i}^\circ$")
pyplot.xlabel("Time (s)")
pyplot.ylabel("Position (rad)")
pyplot.legend()

pyplot.show()

```



Here when the driving force is higher we see that the pendulum is rotating repeatedly through  $2\pi$ rad with variation between the different  $\theta$  values. When the driving force is lower we see an initial difference between the oscillations as a result of the differing initial conditions, however they each stabilize to a common steady-state after some relatively short interval.

```
(b) import numpy as np
import matplotlib.pyplot as pyplot

g = 9.8

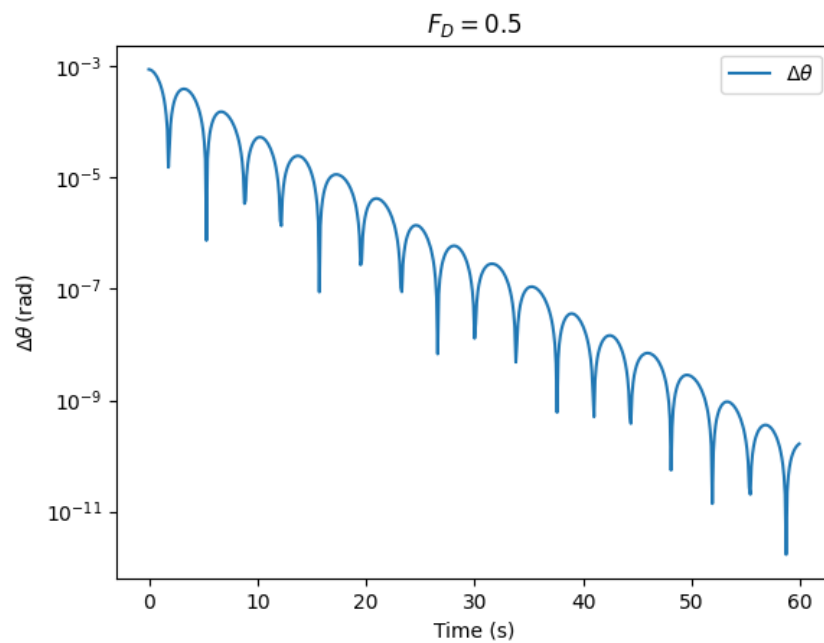
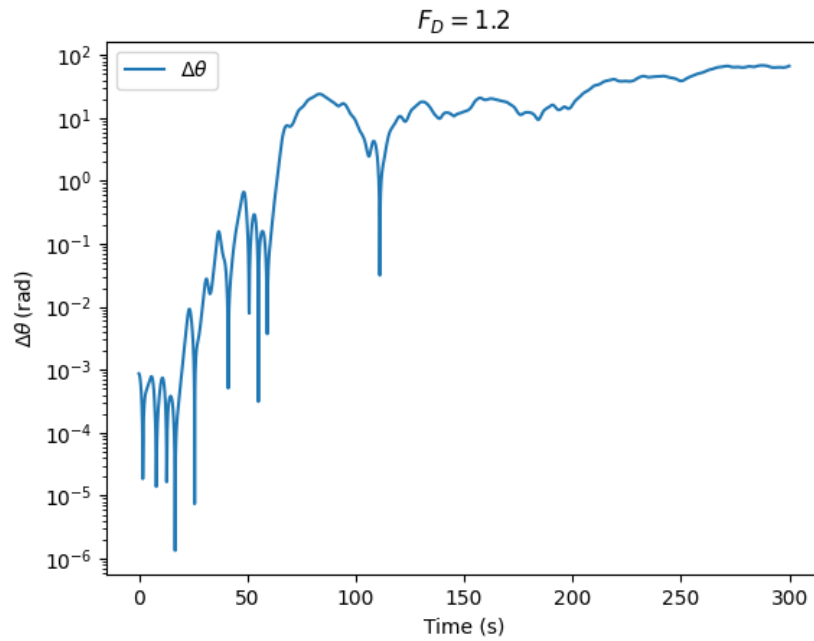
def euler_chromer_nonlinear(theta_0: float,
                             w_0: float,
                             l: float,
                             q: float,
                             F_d: float,
                             W: float,
                             t_vals: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    theta = np.full_like(t_vals, np.deg2rad(theta_0))
    omega = np.full_like(t_vals, w_0)
    dt = t_vals[1]-t_vals[0]
    for t_i in range(len(t_vals) - 1):
        omega[t_i+1] = omega[t_i] + \
            (-(g/l)*np.sin(theta[t_i])-q *
             omega[t_i] + F_d*np.sin(W*t_vals[t_i]))*dt
        theta[t_i+1] = theta[t_i]+omega[t_i+1]*dt

    return theta

t = np.arange(0, 60, 0.1)
pyplot.figure(0)
pyplot.title(r"$F_D=0.5$")
theta_0 = euler_chromer_nonlinear(10, 0, g, 0.5, 0.5, 2/3, t)
theta_1 = euler_chromer_nonlinear(10.05, 0, g, 0.5, 0.5, 2/3, t)
pyplot.yscale("log")
pyplot.plot(t, np.abs(theta_0-theta_1), label=r"$\Delta\theta$")
pyplot.xlabel("Time (s)")
pyplot.ylabel(r"$\Delta\theta$, \text{(rad)}$")
pyplot.legend()

t = np.arange(0, 300, 0.1)
pyplot.figure(1)
pyplot.title(r"$F_D=1.2$")
theta_0 = euler_chromer_nonlinear(10, 0, g, 0.5, 1.2, 2/3, t)
theta_1 = euler_chromer_nonlinear(10.05, 0, g, 0.5, 1.2, 2/3, t)
pyplot.yscale("log")
pyplot.plot(t, np.abs(theta_0-theta_1), label=r"$\Delta\theta$")
pyplot.xlabel("Time (s)")
pyplot.ylabel(r"$\Delta\theta$, \text{(rad)}$")
pyplot.legend()

pyplot.show()
```



Here we see that when the driving force is high enough to overcome whatever “opposing” force is acting on the system even a very slight variation in initial conditions is sufficient to cause “chaotic” behaviour. When the driving force is too weak, we see the opposite case. There is an initial difference (due to the differing initial conditions) which decreases towards a small value of  $\Delta\theta$ .

```
(c) import numpy as np
import matplotlib.pyplot as pyplot

g = 9.8
t = np.arange(0, 300, 0.1)

def euler_chromer_nonlinear(theta_0: float,
```

```

        w_0: float,
        l: float,
        q: float,
        F_d: float,
        W: float,
        t_vals: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    theta = np.full_like(t_vals, np.deg2rad(theta_0))
    omega = np.full_like(t_vals, w_0)
    dt = t_vals[1]-t_vals[0]
    for t_i in range(len(t_vals) - 1):
        omega[t_i+1] = omega[t_i] + \
            (-(g/l)*np.sin(theta[t_i])-q *
             omega[t_i] + F_d*np.sin(W*t_vals[t_i]))*dt
        theta[t_i+1] = theta[t_i]+omega[t_i+1]*dt

    return theta, omega

pyplot.figure(0)
pyplot.title(r"$F_D=0.5$")
theta_0, omega_0 = euler_chromer_nonlinear(10, 0, g, 0.5, 0.5, 2/3, t)

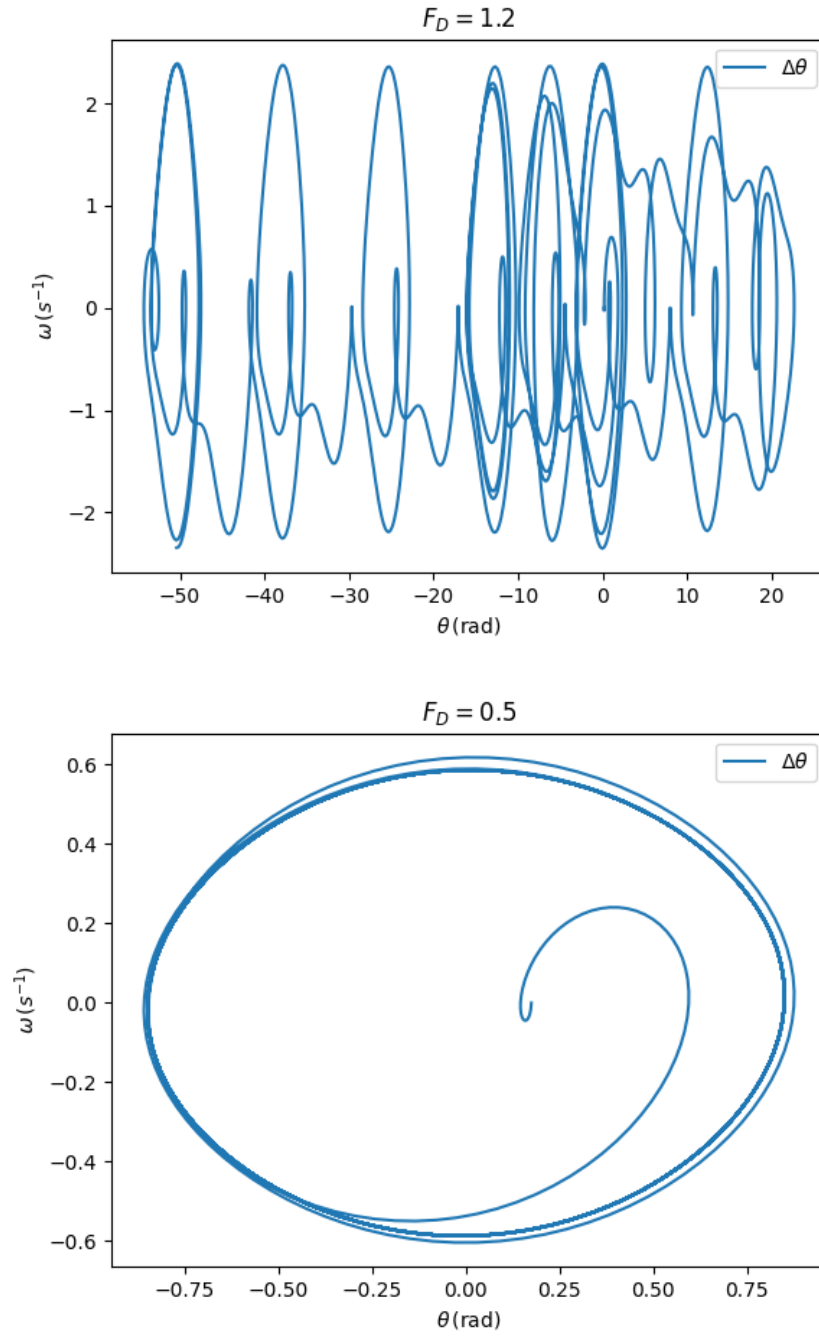
pyplot.plot(theta_0, omega_0, label=r"$\Delta\theta$")
pyplot.ylabel(r"$\omega \, (s^{-1})$")
pyplot.xlabel(r"$\theta \, , \text{(rad)}$")
pyplot.legend()

pyplot.figure(1)
pyplot.title(r"$F_D=1.2$")
theta_0, omega_0 = euler_chromer_nonlinear(10, 0, g, 0.5, 1.2, 2/3, t)

pyplot.plot(theta_0, omega_0, label=r"$\Delta\theta$")
pyplot.ylabel(r"$\omega \, (s^{-1})$")
pyplot.xlabel(r"$\theta \, , \text{(rad)}$")
pyplot.legend()

pyplot.show()

```



As expected from parts a and b, the higher driving force case is able to oscillate with much greater speed (and therefore can rotate through a full  $2\pi\text{rad}$  several times). In the case of a weaker driving force we see, again, an initial blip where  $\omega = 0$  and  $\theta = 10^\circ$  but the oscillator quickly settles into a steady state where  $\theta$  and  $\omega$  oscillate between two values.