Pendulum

May 14, 2023

1 Motion of a Pendulum

This notebook uses the Euler-Cromer method to model the motion of a pendulum.

1.1 Implementation: No Friction or Driving Force

To model the motion of a pendulum without taking friction or driving force into account, we perform the following steps:

- Define the relevant parameters: $l = 1$ m, mass $m = 1$ kg, $\theta_i = 10$ degrees, $n = 250$ points, and time interval $\Delta t = 0.04$ s.
- Define the arrays to store the time, the angular velocity, the angle of the pendulum, and the energy.
- Write the loop to solve the following differential equations without friction using Euler method:

$$\frac{d\omega}{dt} = -\frac{g}{l}\sin(\theta)$$
$$\frac{d\theta}{dt} = \omega$$

- Assume that the energy of the pendulum is given by:

$$E = \frac{1}{2}ml^2\omega^2 + mgl(1 - \cos(\theta))$$

- Graph the solutions θ vs. t, ω vs. t, and energy vs. t.

[ ]:
```
import math

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

num_points = 250
time_step = 0.04
l = 1
m = 1
g = 9.80665
initial_theta_degrees = 10
```

```
t = np.empty(num_points)
omega = np.empty(num_points)
theta = np.empty(num_points)
energy = np.empty(num_points)

t[0] = 0
omega[0] = 0
theta[0] = initial_theta_degrees * math.pi / 180
energy[0] = m * l * l * omega[0] * omega[0] / 2 + m * g * l * (1 - math.
 ↪cos(theta[0]))

for i in range(1, num_points):
    t[i] = i * time_step
    omega[i] = omega[i - 1] - g * theta[i - 1] * time_step / l
    theta[i] = theta[i - 1] + omega[i] * time_step
    energy[i] = m * l * l * pow(omega[i - 1], 2) / 2 + m * g * l * (1 - math.
 ↪cos(theta[i - 1]))

data = np.ndarray(shape=(num_points, 4), dtype=float)
for i in range(num_points):
    data[i][0] = t[i]
    data[i][1] = omega[i]
    data[i][2] = theta[i]
    data[i][3] = energy[i]
dataframe = pd.DataFrame(data, columns = ["t", "omega", "theta", "energy"])
dataframe
```

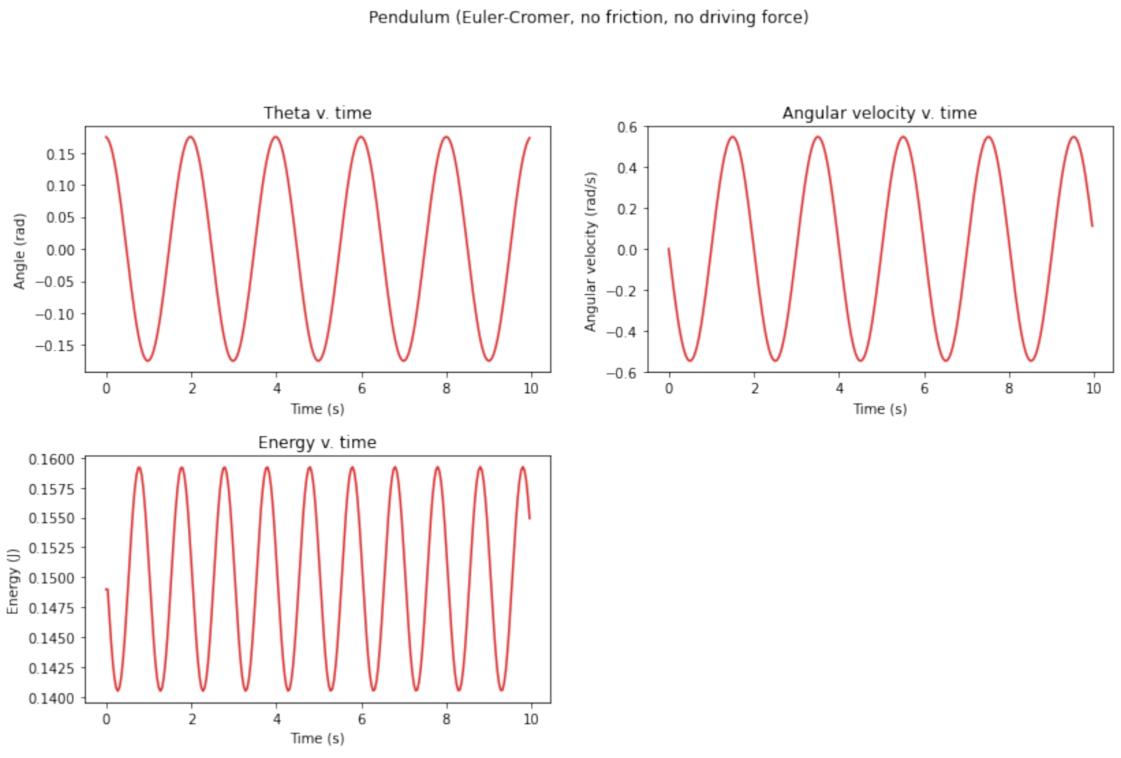
[ ]:

```
        t     omega     theta    energy
0    0.00  0.000000  0.174533  0.148985
1    0.04 -0.068463  0.171794  0.148985
2    0.08 -0.135852  0.166360  0.146701
3    0.12 -0.201110  0.158316  0.144618
4    0.16 -0.263212  0.147787  0.142863
..    …        …         …         …
245  9.80  0.355539  0.139860  0.159238
246  9.84  0.300677  0.151887  0.158960
247  9.88  0.241097  0.161531  0.158104
248  9.92  0.177734  0.168640  0.156724
249  9.96  0.111582  0.173103  0.154912

[250 rows x 4 columns]
```

[ ]:

```
fig, axs = plt.subplots(2, 2, sharex=False, sharey=False, figsize=(12, 8))
fig.suptitle("Pendulum (Euler-Cromer, no friction, no driving force)")
fig.tight_layout(pad=4)
```

```
axs[0, 0].plot(t, theta, "tab:red")
axs[0, 0].set_title("Theta v. time")
axs[0, 0].set_xlabel('Time (s)')
axs[0, 0].set_ylabel('Angle (rad)')

axs[0, 1].plot(t, omega, "tab:red")
axs[0, 1].set_title("Angular velocity v. time")
axs[0, 1].set_xlabel('Time (s)')
axs[0, 1].set_ylabel('Angular velocity (rad/s)')

axs[1, 0].plot(t, energy, "tab:red")
axs[1, 0].set_title("Energy v. time")
axs[1, 0].set_xlabel('Time (s)')
axs[1, 0].set_ylabel('Energy (J)')

axs[1, 1].axis('off')

plt.show()
```



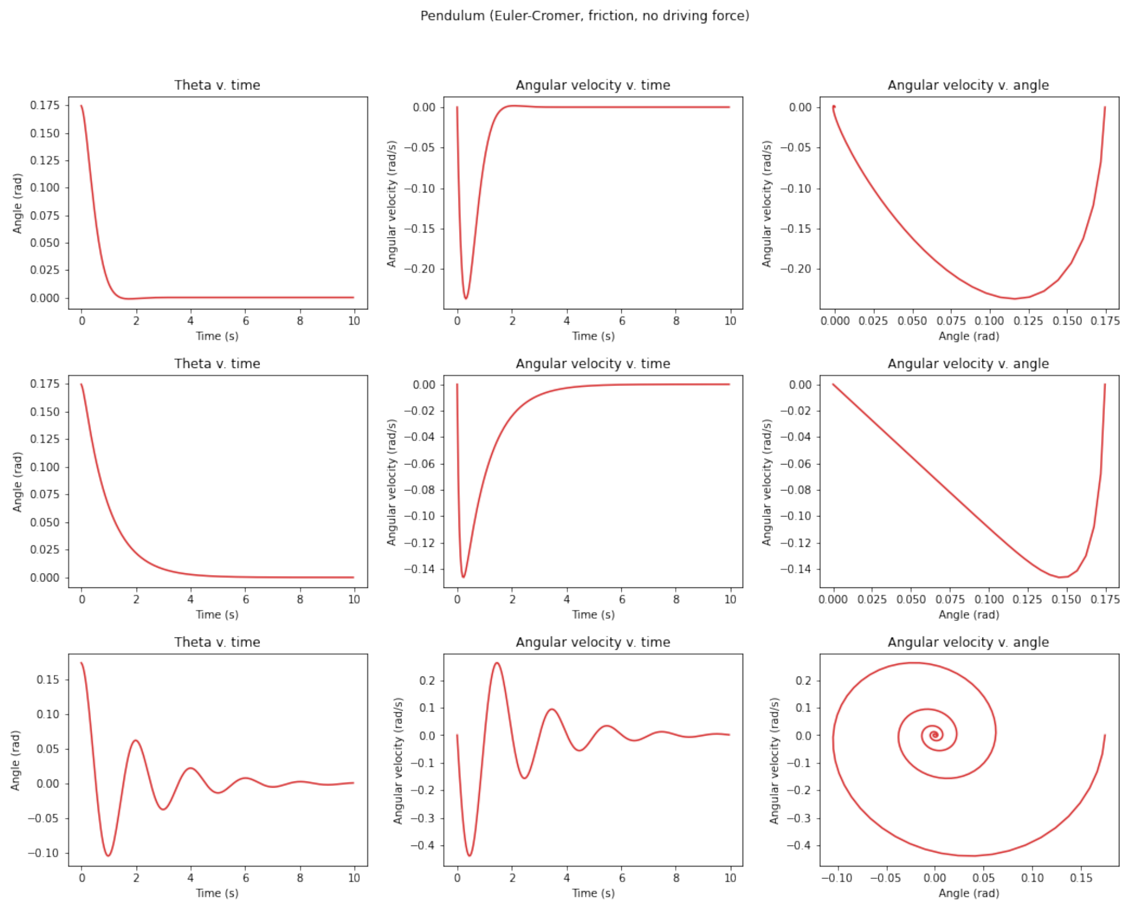
1.2 Implementation: Pendulum with Friction

Now assume that the pendulum's motion can be modeled as the following:

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

where we might set the damping $q = 5$ (critically-damped), $q = 10$ (over-damped), or $q = 1$ (under-damped). Plot θ vs. t, ω vs. t, and ω vs. θ for these values.

[ ]:
```
q_critical = 5
q_overdamped = 10
q_underdamped = 1

t[0] = 0
omega[0] = 0
theta[0] = initial_theta_degrees * math.pi / 180

fig, axs = plt.subplots(3, 3, sharex=False, sharey=False, figsize=(15, 12))
fig.suptitle("Pendulum (Euler-Cromer, friction, no driving force)")
fig.tight_layout(pad=4)

for q_idx, q in enumerate([q_critical, q_overdamped, q_underdamped]):
    for i in range(1, num_points):
        t[i] = i * time_step
        omega[i] = omega[i - 1] - g * math.sin(theta[i - 1]) * time_step / l -␣
 ↪q * omega[i - 1] * time_step
        theta[i] = theta[i - 1] + omega[i] * time_step

    axs[q_idx, 0].plot(t, theta, "tab:red")
    axs[q_idx, 0].set_title("Theta v. time")
    axs[q_idx, 0].set_xlabel('Time (s)')
    axs[q_idx, 0].set_ylabel('Angle (rad)')

    axs[q_idx, 1].plot(t, omega, "tab:red")
    axs[q_idx, 1].set_title("Angular velocity v. time")
    axs[q_idx, 1].set_xlabel('Time (s)')
    axs[q_idx, 1].set_ylabel('Angular velocity (rad/s)')

    axs[q_idx, 2].plot(theta, omega, "tab:red")
    axs[q_idx, 2].set_title("Angular velocity v. angle")
    axs[q_idx, 2].set_xlabel('Angle (rad)')
    axs[q_idx, 2].set_ylabel('Angular velocity (rad/s)')

plt.show()
```



1.3 Implementation: Pendulum with Friction and Driving Force

Now assume that the pendulum's motion can be modeled as the following:

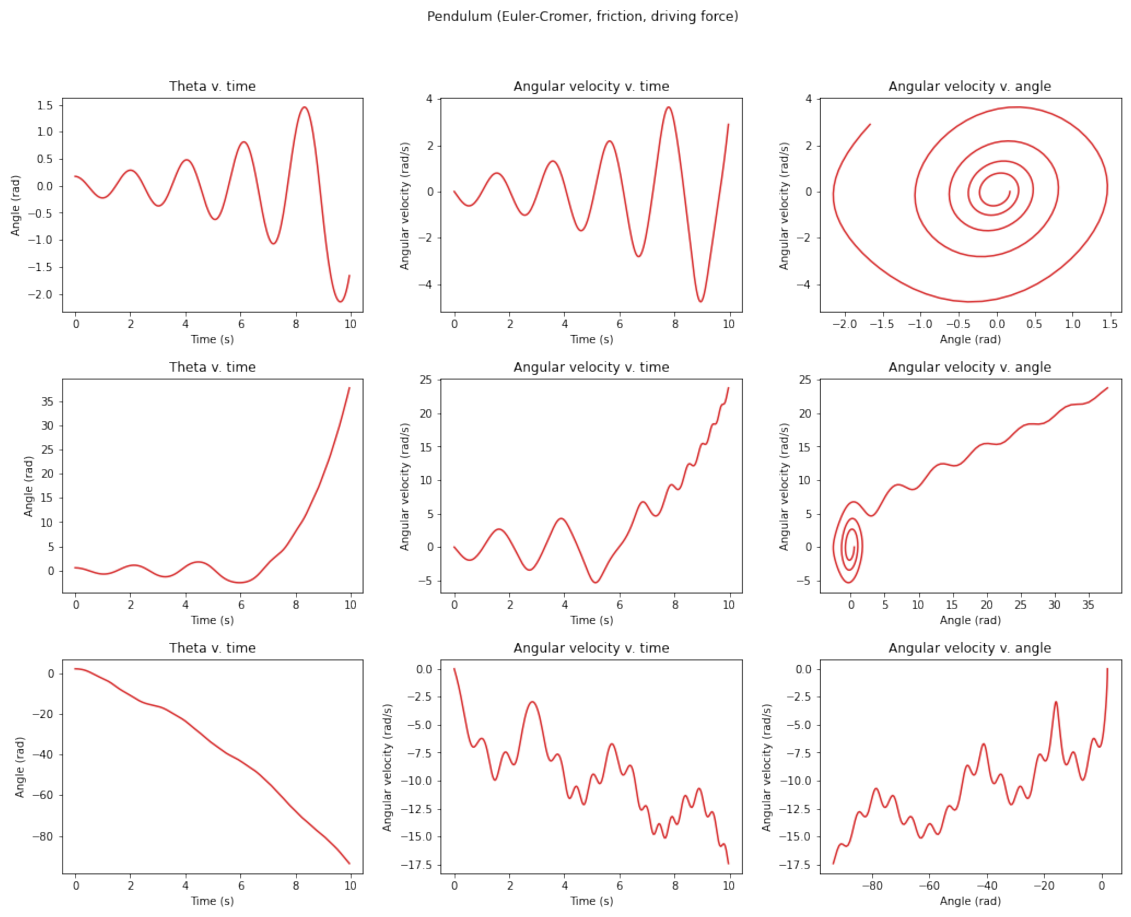
$$\begin{aligned}\frac{d\omega}{dt} &= -\frac{g}{l}\sin(\theta) - q\frac{d\theta}{dt} + F\sin(\Omega_D t)\\ \frac{d\theta}{dt} &= \omega\end{aligned}$$

Assume that $\Omega_D = 2.13$ Hz. We might have the following cases: - $\theta_i = 10$ deg, $F = 0$ N, and $q = 0.5$. - $\theta_i = 30$ deg, $F = 0.5$ N, and $q = 0.5$. - $\theta_i = 117$ deg, $F = 5.2$ N, and $q = 0.1$.

We generate the same graphs as the friction case in these three scenarios.

```
[ ]: omega_d = 2.13
     options = [
         { "initial_theta_degrees": 10, "force": 0, "q": 0.5 },
         { "initial_theta_degrees": 30, "force": 0.5, "q": 0.5 },
         { "initial_theta_degrees": 117, "force": 5.2, "q": 0.1 },
     ]
```

```
fig, axs = plt.subplots(3, 3, sharex=False, sharey=False, figsize=(15, 12))
fig.suptitle("Pendulum (Euler-Cromer, friction, driving force)")
fig.tight_layout(pad=4)

for idx, option in enumerate(options):
    t[0] = 0
    omega[0] = 0
    theta[0] = option["initial_theta_degrees"] * math.pi / 180

    for i in range(1, num_points):
        t[i] = i * time_step
        omega[i] = omega[i - 1] - (g * math.sin(theta[i - 1]) / l - option["q"]␣
 ↪* omega[i - 1] + option["force"] * math.sin(omega_d * t[i - 1])) * time_step
        theta[i] = theta[i - 1] + omega[i] * time_step

    axs[idx, 0].plot(t, theta, "tab:red")
    axs[idx, 0].set_title("Theta v. time")
    axs[idx, 0].set_xlabel('Time (s)')
    axs[idx, 0].set_ylabel('Angle (rad)')

    axs[idx, 1].plot(t, omega, "tab:red")
    axs[idx, 1].set_title("Angular velocity v. time")
    axs[idx, 1].set_xlabel('Time (s)')
    axs[idx, 1].set_ylabel('Angular velocity (rad/s)')

    axs[idx, 2].plot(theta, omega, "tab:red")
    axs[idx, 2].set_title("Angular velocity v. angle")
    axs[idx, 2].set_xlabel('Angle (rad)')
    axs[idx, 2].set_ylabel('Angular velocity (rad/s)')

plt.show()
```



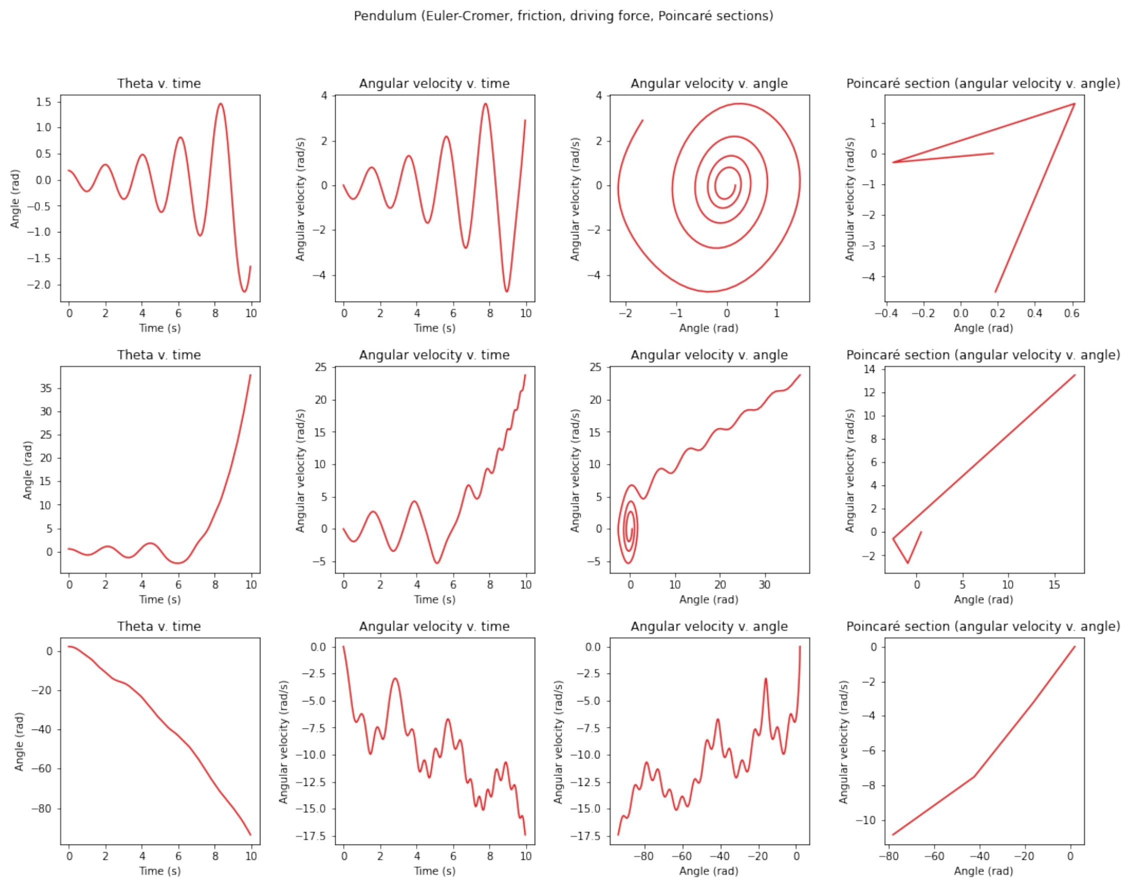
1.4 Implementation: Pendulum with Friction, Driving Force, and the Poincaré Section

Sometimes, chaos will result in extremely erratic plots with no apparent correlation between quantities. The Poincaré section can be helpful in these cases by plotting only a subset of points. In this case, we only plot the points in phase with the driving force: the ω vs. θ only at times that are multiples of the period of the driving force. This is similar to taking a picture of the pendulum at every $2\pi/\Omega_D$ and recording the ω and θ each time. If the pendulum is more chaotic, then the Poincaré section will have points distributed in the phase space.

[ ]:
```
poincare_theta = np.empty(num_points)
poincare_omega = np.empty(num_points)

fig, axs = plt.subplots(3, 4, sharex=False, sharey=False, figsize=(15, 12))
fig.suptitle("Pendulum (Euler-Cromer, friction, driving force, Poincaré␣
  ↪sections)")
fig.tight_layout(pad=4)

for idx, option in enumerate(options):
```

```
t[0] = 0
omega[0] = 0
theta[0] = option["initial_theta_degrees"] * math.pi / 180
num_poincare_points = 0

for i in range(1, num_points):
    t[i] = i * time_step
    omega[i] = omega[i - 1] - (g * math.sin(theta[i - 1]) / l - option["q"]␣
↪* omega[i - 1] + option["force"] * math.sin(omega_d * t[i - 1])) * time_step
    theta[i] = theta[i - 1] + omega[i] * time_step

period = 2 * math.pi / omega_d
iter = 0
while iter <= t[num_points - 1]:
    # Find the time closest to the current multiple of period (iter)
    best_distance = 1000
    best_index = -1
    for i in range(num_points):
        if abs(iter - t[i]) < best_distance:
            best_distance = abs(iter - t[i])
            best_index = i
    poincare_theta[num_poincare_points] = theta[best_index]
    poincare_omega[num_poincare_points] = omega[best_index]
    num_poincare_points = num_poincare_points + 1
    iter = iter + period

poincare_theta = np.resize(poincare_theta, num_poincare_points)
poincare_omega = np.resize(poincare_omega, num_poincare_points)

axs[idx, 0].plot(t, theta, "tab:red")
axs[idx, 0].set_title("Theta v. time")
axs[idx, 0].set_xlabel('Time (s)')
axs[idx, 0].set_ylabel('Angle (rad)')

axs[idx, 1].plot(t, omega, "tab:red")
axs[idx, 1].set_title("Angular velocity v. time")
axs[idx, 1].set_xlabel('Time (s)')
axs[idx, 1].set_ylabel('Angular velocity (rad/s)')

axs[idx, 2].plot(theta, omega, "tab:red")
axs[idx, 2].set_title("Angular velocity v. angle")
axs[idx, 2].set_xlabel('Angle (rad)')
axs[idx, 2].set_ylabel('Angular velocity (rad/s)')

axs[idx, 3].plot(poincare_theta, poincare_omega, "tab:red")
axs[idx, 3].set_title("Poincaré section (angular velocity v. angle)")
axs[idx, 3].set_xlabel('Angle (rad)')
```

```
    axs[idx, 3].set_ylabel('Angular velocity (rad/s)')

plt.show()
```


Pendulum (Euler-Cromer, friction, driving force, Poincaré sections)
Theta v. time
Angular velocity v. time
Angular velocity v. angle
Poincaré section (angular velocity v. angle)
Angle (rad)
Angular velocity (rad/s)
Time (s)
Angle (rad)