```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from scipy.integrate import solve_ivp

speed_factor = 2

def plot_double_pendulum(θ1, θ2, l1, l2, ax, line_color='red',
line_width=3, marker_color='blue', marker_size=10):
    # Calculate x and y coordinates from θ1 and θ2 for the pendulum
ends
    x1 = l1 * np.sin(θ1)
    y1 = -l1 * np.cos(θ1)
    x2 = x1 + l2 * np.sin(θ2)
    y2 = y1 - l2 * np.cos(θ2)

    ax.plot([0, x1, x2], [0, y1, y2], lw=line_width, color=line_color)
    ax.scatter([x1, x2], [y1, y2], color=marker_color, s=marker_size)
    ax.set_aspect('equal')
    ax.set_xlim(-0.5, 0.5)
    ax.set_ylim(-0.5, 0.5)
    ax.axis('off')

def animate_double_pendulum(sol, l1, l2, timestep=0.05, fps=30):
    # Create frames for the solution plot
    times = np.arange(sol.t[0], sol.t[-1], timestep)
    fig, ax = plt.subplots()

    def update(frame):
        ax.cla()
        θ1, θ2 = sol.sol(frame)[:2]
        plot_double_pendulum(θ1, θ2, l1, l2, ax)

    anim = FuncAnimation(fig, update, frames=times, interval=(1000 /
fps) * speed_factor)
    plt.show()
    return anim

def double_pendulum(t, u, p):
    θ1, θ2, ω1, ω2 = u
    m1, m2, l1, l2, g = p
    delta = θ2 - θ1
    den1 = (m1 + m2) * l1 - m2 * l1 * np.cos(delta) * np.cos(delta)
```

```
    den2 = (l2 / l1) * den1
    dθ1dt = ω1
    dθ2dt = ω2
    dω1dt = (m2 * l1 * ω1 * ω1 * np.sin(delta) * np.cos(delta) +
             m2 * g * np.sin(θ2) * np.cos(delta) +
             m2 * l2 * ω2 * ω2 * np.sin(delta) -
             (m1 + m2) * g * np.sin(θ1)) / den1
    dω2dt = (-m2 * l2 * ω2 * ω2 * np.sin(delta) * np.cos(delta) +
             (m1 + m2) * (g * np.sin(θ1) * np.cos(delta) -
                          l1 * ω1 * ω1 * np.sin(delta) -
                          g * np.sin(θ2))) / den2
    return [dθ1dt, dθ2dt, dω1dt, dω2dt]

# Initial conditions
θ1 = 5.01 * np.pi
θ2 = np.pi / 4.5
ω1 = 0.0
ω2 = 0.0
u0 = [θ1, θ2, ω1, ω2]

# Parameters
m1 = 0.21347061294937527
m2 = 0.13320228970158902
l1 = 0.2
l2 = 0.12
g = 9.81
p = [m1, m2, l1, l2, g]

# Time span for animation
tspan = (0, 10)

# Solve ODE
sol = solve_ivp(double_pendulum, tspan, u0, args=(p,),
dense_output=True, rtol=1e-6, atol=1e-6)

# Animate double pendulum
anim = animate_double_pendulum(sol, l1, l2)

# Plot θ1 and θ2 over time
plt.figure()
plt.plot(sol.t, sol.y[0], label='θ1')
plt.plot(sol.t, sol.y[1], label='θ2')
plt.legend()
```

```
plt.xlabel('Time')
plt.ylabel('Angle (rad)')
plt.show()









def determine_x2_y2(θ1, θ2, l1, l2):

    x1 = l1 * np.sin(θ1)
    y1 = -l1 * np.cos(θ1)
    x2 = x1 + l2 * np.sin(θ2)
    y2 = y1 - l2 * np.cos(θ2)

    return x2, y2


x2, y2 = determine_x2_y2(sol.y[0],  sol.y[1], l1, l2)

# # Plot x2 and y2 over time
# plt.figure()
# plt.plot(sol.t, x2, label='x2')
# plt.plot(sol.t, y2, label='y2')
# plt.legend()
# plt.xlabel('Time')
# plt.ylabel('Coordinates')
# plt.show()




data = np.loadtxt(r'C:\Users\fiepm\OneDrive\studie\DEF\test.csv',
skiprows = 1, delimiter = ';')  #eerste meting
x2_m1 = data[:,0]
y2_m1 = data[:,1]

# data = np.loadtxt('....csv', skiprows = 1, delimiter = ';')  #tweede
meting
```

```
# x2_m1 = data[:,0]
# y2_m1 = data[:,1]



# Plot (x2,y2) trajectory
plt.figure()
plt.plot(x2, y2, label = 'trajectory model')
# plt.plot(x2_m1, y2_m1, label = 'trajectory measurement 1')
# plt.plot(x2_m2, y2_m2, label = 'trajectory measurement 2')
plt.plot(x2[0], y2[0], 'o', label='initial condition')
plt.legend()
plt.xlabel('x2')
plt.ylabel('y2')
plt.show()
```