Eric Du | Physics 105
HW 11 | **Analytic Mechanics** | April 22, 2023

Collaborators

I worked with **Andrew Binder** and **Adarsh Iyer** to complete this assignment.

Problem 1

Determine the Hamiltonian and Hamilton's equations for the double Atwood machine given in Homework 2 Problem 4.

Solution: For this problem, let the height of masses 1 and 2 be m_1 and m_2, and the height of the pulley be X. From the previous homeworks, we know that:

$$\mathcal{L} = T - U = \frac{1}{2}m_1\dot{x}_1^2 + \frac{1}{2}m_2\dot{x}_2^2 + \frac{1}{2}m\left(\frac{\dot{x}_1+\dot{x}_2}{2}\right)^2 - m_1gx_1 - m_2gx_2 + Mg\left(\frac{x_1+x_2}{2}\right)$$

Then, since we're in a natural coordinate system, we know that $\mathcal{H} = T + U$, therefore:

$$\mathcal{H} = T + U = \frac{1}{2}m_1\dot{x}_1^2 + \frac{1}{2}m_2\dot{x}_2^2 + \frac{1}{2}m\left(\frac{\dot{x}_1+\dot{x}_2}{2}\right)^2 + m_1gx_1 + m_2gx_2 - Mg\left(\frac{x_1+x_2}{2}\right)$$

Then, using $p_i = \frac{\partial\mathcal{L}}{\partial\dot{q}_i}$, we get the system of equations:

$$\begin{aligned} p_1 &= m_1\dot{x}_1 + \frac{M}{4}(\dot{x}_1+\dot{x}_2) \\ p_2 &= m_2\dot{x}_2 + \frac{M}{4}(\dot{x}_1+\dot{x}_2) \end{aligned}$$

Multiplying these equations by 4 so we don't have to deal with fractions, we get:

$$\begin{aligned} 4p_1 &= \dot{x}_1(4m_1+M) + M\dot{x}_2 \\ 4p_2 &= \dot{x}_2(4m_2+M) + M\dot{x}_1 \end{aligned}$$

Solving this system of equations, we get:

$$\begin{aligned} \dot{x}_1 &= \frac{p_1(4m_2+M) - Mp_2}{4m_2m_1 + M(m_2+m_1)} \\ \dot{x}_2 &= \frac{p_2(4m_1+M) - Mp_1}{4m_1m_2 + M(m_1+m_2)} \end{aligned}$$

Therefore, the Hamiltonian becomes:

$$\begin{aligned} H = \frac{1}{2}m_1\left[\frac{p_1(4m_2+M)-Mp_2}{4m_2m_1+M(m_2+m_1)}\right]^2 + \frac{1}{2}m_2\left[\frac{p_2(4m_1+M)-Mp_1}{4m_1m_2+M(m_1+m_2)}\right]^2 + \\ \frac{1}{2}M\left[\frac{4(p_1m_2+p_2m_1)-M(p_2+p_1)}{2(4m_2m_1+M(m_1+m_2))}\right]^2 + m_1gx_1 + m_2gx_2 - Mg\left(\frac{x_1+x_2}{2}\right) \end{aligned}$$

Now, Hamilton's equations means that we have to find $\frac{\partial \mathcal{H}}{\partial p_i}$. Therefore:

$$\dot{x}_1 = \frac{\partial \mathcal{H}}{\partial p_1} = m_1 \left[\frac{p_1(4m_2+M) - Mp_2}{4m_2m_1 + M(m_2+m_1)}\right] \cdot \frac{4m_2+M}{4m_2m_1+M(m_1+m_2)} - \\ m_2\left[\frac{p_2(4m_1+M)-Mp_1}{4m_1m_2+M(m_1+m_2)}\right] \cdot \frac{M}{4m_2m_1+M(m_1+m_2)} + \\ M\left[\frac{4(p_1m_2+p_2m_1 - M(p_2+p_1)}{2(4m_2m_1+M(m_1+m_2))}\right] \cdot \frac{4m_2-M}{2(4m_2m_1+M(m_1+m_2))}$$

With the power of Mathematica, this simplifies to:

$$\dot{x}_1 = \frac{4m_2p_1 + M(p_1-p_2)}{4m_1m_2+M(m_1+m_2)}$$

also, the equations here are symmetric in the sense that for $\dot{x}_2$, it's going to be the expression for x_1 except we change every $1 \to 2$ and $2 \to 1$, so therefore:

$$\dot{x}_2 = \frac{4m_1p_2 + M(p_2-p_1)}{4m_1m_2+M(m_1+m_2)}$$

To get the other two equations we can use the relation $\dot{p}_i = \frac{\partial \mathcal{H}}{\partial q}$:

$$\dot{p}_1 = \left(\frac{M}{2} - m_1\right) g x_1$$
$$\dot{p}_2 = \left(\frac{M}{2} - m_2\right) g x_2$$

□

Problem 2

Consider a simple plane pendulum of mass m and length ℓ. After the pendulum is set into motion, the length of the string is shortened at a constant rate.

$$\frac{\mathrm{d}\ell}{\mathrm{d}t} = -\alpha$$

The suspension point remains fixed. Determine the both Lagrangian and Hamiltonian functions. Compare the Hamiltonian with $E = T + U$, and discuss conservation of energy for this system.

Solution: We write the kinetic energy as $T = \frac{1}{2}m\ell^2\dot{\phi}^2 + \frac{1}{2}m\alpha^2$, and the potential energy is $U = -mg\ell\cos\phi$, so therefore the Lagrangian becomes:

$$\mathcal{L} = \frac{1}{2}m\ell^2\dot{\phi}^2 + \frac{1}{2}m\alpha^2 + mg\ell\cos\alpha$$

From here, we can get the generalized momenta using $p_i = \frac{\partial\mathcal{L}}{\partial\dot{q}_i}$:

$$p_l = \frac{\partial\mathcal{L}}{\partial\dot{\ell}} = 0$$
$$p_\phi = \frac{\partial\mathcal{L}}{\partial\dot{\phi}} = m\ell^2\dot{\phi}$$

So now, we can use $\mathcal{H} = \sum_i p_i q_i - \mathcal{L}$ to get:

$$\mathcal{H} = p_\phi\dot{\phi} - \left(\frac{1}{2}m\ell^2\dot{\phi}^2 + \frac{1}{2}m\alpha^2 + mgl\cos\alpha\right) = \frac{1}{2}m\ell^2\dot{\phi}^2 - \frac{1}{2}m\alpha^2 - mgl\cos\phi$$

Here, we can see that the Hamiltonian isn't the total energy, which is given by $T + U = \frac{1}{2}m\alpha^2 + \frac{1}{2}m\ell^2\dot{\phi}^2 - mgl\cos\phi$. This is because the Hamiltonian $\mathcal{H}$ has explicit time dependence, namely in the parameter l and the fact that l changes in time as $\ell(t) = \ell_0 - \alpha t$, so thus $\frac{\partial\mathcal{H}}{\partial t} \neq 0$. $\square$

Problem 3

Consider a particle of mass m constrained to move on a frictionless cylinder of radius R, described in cylindrical coordinates by the surface $\rho = R$. The mass is attached to a spring (spring constant k and equilibrium length zero) whose other end is fastened at the origin. Using cylindrical coordinates, (ρ, ϕ, z), find the Hamiltonian H. Then write down and solve Hamilton's equations and describe the motion.

Solution: Since cylindrical coordinates are natural, we know that the Hamiltonian is just going to be the kinetic plus potential energy. Setting the axis to be the origin, we can now write down the kinetic and potential energies in cylindrical coordinates:

$$T = \frac{m}{2}(\dot{\rho}^2 + \rho^2\dot{\phi}^2 + \dot{z}^2) + \frac{1}{2}(R^2\dot{\phi}^2 + z^2)$$
$$V = \frac{1}{2}kr^2 = \frac{1}{2}k(R^2 + z^2)$$

We can cancel the $\dot{\rho}$ terms since the mas sis constrained to move on the cylinder. Therefore, we have:

$$\mathcal{H} = T + V = \frac{m}{2}(R^2\dot{\phi}^2 + \dot{z}^2) + \frac{k}{2}(R^2 + z^2)$$

Computing the generalized momenta using $p_i = \frac{\mathrm{d}T}{\mathrm{d}q_i}$, since we're working in natural coordinates :

$$p_z = \frac{\mathrm{d}T}{\mathrm{d}\dot{z}} = m\dot{z}$$
$$p_\phi = \frac{\mathrm{d}T}{\mathrm{d}\dot{\phi}} = mR^2\dot{\phi}$$

So Hamilton's equation then becomes:

$$\mathcal{H} = \frac{m}{2}\left(R^2\frac{p_\phi^2}{(mR^2)^2} + \frac{p_z^2}{m^2}\right) + \frac{k}{2}(R^2 + z^2) = \frac{p_\phi^2}{2mR^2} + \frac{p_z^2}{2m} + \frac{k}{2}(R^2 + z^2)$$

Finally, we can write Hamilton's equations:

$$\dot{\phi} = \frac{\partial \mathcal{H}}{\partial p_\phi} = \frac{p_\phi}{2mR^2}$$
$$\dot{z} = \frac{\partial \mathcal{H}}{\partial p_z} = \frac{p_z}{m}$$
$$\dot{p}_\phi = -\frac{\partial \mathcal{H}}{\partial \phi} = 0$$
$$\dot{p}_z = -\frac{\partial \mathcal{H}}{\partial z} = -kz$$

These then give us our equations of motion. Since $\dot{p}_\phi = 0$, this implies that p_ϕ is a constant, and thus $\dot{\phi}$, which is defined in terms of p_ϕ, is also a constant. In the z direction, we have

$$\dot{z} = \frac{p_z}{m} \implies \ddot{z} = -\frac{k}{m}z$$

And so we get $z(t) = A\cos(\omega t - \delta)$, where A, δ are some constants given by initial conditions.

In terms of the motion of the particle, we see that it oscillates sinusoidally in the z direction, and moves around in the $\hat{\phi}$ direction with a uniform velocity. This makes intuitive sense as well, since the spring force always acts perpendicular to $\hat{\phi}$, so we don't expect any change in velocity in that direction. The same cannot be said for the z direction, which is why we sinusoidal oscillations. $\square$

Problem 4

Consider the modified Atwood machine shown in Figure 13.11. The two weights on the left have equal mass m and are connected by a massless spring of force constant k. The weight on the right has mass $M = 2m$, and the pulley is massless and frictionless. The coordinate x is the extension of the spring from its equilibrium length; that is, the length of the spring is $l_e + x$, where l_e is the equilibrium length (with all the weights in position M held stationary).

a) Show that the total potential energy (spring plus gravitational) is just $U = \frac{1}{2}kx^2$ (plus a constant that can be taken to zero).

Solution: Using the position of the big wheel as zero potential energy, we can write the gravitational potential energy in the system as:

$$U_g = -mgy - mg(y + l_e + x) - Mg(L - y) = mg(y + y + l_e + x + 2(L - y) = -mgx + \text{const.}$$

Then, to express spring potential energy, let l_0 denote the natural length of the spring. From balancing of forces, we see that:

$$k(le - l_0) = mg$$

Then, we can now calculate the spring potential energy:

$$U_s = \frac{1}{2}k(x + le - l_0)^2 = \frac{1}{2}k\left(x + \frac{mg}{k}\right)^2 = \frac{1}{2}kx^2 + mgx + \frac{1}{2}k^3$$

Combining the two expressions now:

$$U_g + U_s = \frac{1}{2}kx^2 + mgx + \frac{1}{2}k^3 - mgx - \text{const.}$$

Since $\frac{1}{2}k^3$ is also a constant and the mgx terms cancel, then we see that the total energy in the system is indeed equal to $\frac{1}{2}kx^2$ plus a constant. □

---

b) Find the two momenta conjugate to x and y. Solve for $\dot{x}$ and $\dot{y}$ and write down the Hamiltonian. Show that the coordinate y is ignorable.

Solution: Here, since the coordinate system is natural, we can find p_i by taking $\frac{\mathrm{d}T}{\mathrm{d}\dot{q}_i}$. First, calculating T:

$$T = \frac{1}{2}M\dot{y}^2 + \frac{1}{2}my^2 + \frac{1}{2}m(\dot{x} + \dot{y})^2 = \frac{3}{2}m\dot{y}^2 + \frac{1}{2}m(\dot{x} + \dot{y})^2$$

therefore, we can now get p_i:

$$p_x = \frac{\partial T}{\partial \dot{x}} = m(\dot{x} + \dot{y})$$
$$p_y = \frac{\partial T}{\partial \dot{y}} = 3m\dot{y} + m(\dot{x} + \dot{y})$$

From these equations, we can then backsolve for $\dot{y}$ and $\dot{x} + \dot{y}$

$$\dot{x} + \dot{y} = \frac{p_x}{m}$$
$$\dot{y} = \frac{p_y - p_x}{3m}$$

Therefore, the Hamiltonian written in these coordinates is:

$$\mathcal{H} = T + U = \frac{1}{2}m\left[\frac{p_x^2}{m^2} + \frac{(p_y - p_x)^2}{m^2}\right] + \frac{1}{2}kx^2$$

Since $\frac{\partial \mathcal{H}}{\partial y} = 0$, then this means that y is an ignorable coordinate. □

---

c) Write down the four Hamilton equations and solve them for the following initial conditions: You hold the mass M fixed with the whole system in equilibrium and $y = y_0$. Still holding M fixed, you pull the lower mass m a distance x_0, and at $t = 0$ you let go of both masses. [*Hint:* Write down the initial values x, y and their momenta. You can solve the x equations by combining them into a second-order equation for x. Once you know $x(t)$, you can quickly write down the other three variables.] Describe the motion. In particular, find the frequency with which x oscillates.

Solution: The four Hamilton equations come from just taking derivatives:

$$\begin{aligned}
\dot{x} &= \frac{\partial \mathcal{H}}{\partial p_x} = \frac{1}{3m}(4p_x - p_y) \\
\dot{y} &= \frac{\partial \mathcal{H}}{\partial p_y} = \frac{p_y - p_x}{3m} \\
\dot{p}_x &= -\frac{\partial \mathcal{H}}{\partial x} = -kx \\
\dot{p}_y &= -\frac{\partial \mathcal{H}}{\partial y} = 0
\end{aligned}$$

Since we hold M fixed, then this also means that we're holding the mass on the other side to be fixed as well. This gives the initial conditions: $\dot{x}(0) = 0$ and $\dot{y}(0) = 0$, implying that $p_x(0) = p_y(0) = 0$. Further, from Hamilton's equations, we can find $\ddot{x}$

$$\ddot{x} = \frac{1}{3m}(4\ddot{p}_x - \ddot{p}_y) = \frac{4k}{3m}x$$

This is the same equation of motion as a harmonic oscillator, so therefore x oscillates with frequency $\omega = \sqrt{\frac{4k}{3m}}$. Therefore:

$$x(t) = x_0 \cos(\omega t)$$

As for the other coordinates, since we know that $\dot{p}_x = kx$, then:

$$p_x(t) = \frac{kx_0}{\omega}\sin(\omega t)$$

Finally, we can get $y(t)$ using $\dot{y} = -\frac{p_x}{3m}$:

$$y(t) = \frac{1}{3m}\frac{x_0}{\omega}\cos(\omega t) + y_0$$

□

Problem 5

A spherical pendulum consists of a mass m attached to a massless, rigid rod of length ℓ. The end of the rod opposite the mass can pivot freely (hence "spherical" pendulum) in all directions about some fixed point.

a) Set up the Hamiltonian function (in spherical coordinates!). Check that if $p_\phi = 0$, the result is the same as that for a plane pendulum.

Solution: To set up the Hamiltonian, we again use the fact that our coordinate system is natural, so the Hamiltonian in this case will just be the sum of our potential and kinetic energies. Writing this out in spherical coordinates, we get:

$$T = \frac{1}{2}m(\dot{r}^2 + \ell^2\dot{\theta}^2 + \ell^2\sin^2\theta\dot{\phi}^2) = \frac{m}{2}\left(\ell^2\dot{\theta}^2 + \ell^2\sin^2\theta\dot{\phi}^2\right) V = mg\cos\theta$$

We drop the $\dot{\ell}$ term because the rod is rigid, so there is no change radially. Now writing out the generalized momentum using $p_i = \frac{\mathrm{d}T}{\mathrm{d}\dot{q}_i}$ (again, we can do this because our coordinate system is natural):

$$p_\theta = \frac{\partial T}{\partial\dot{\theta}} = m\ell^2\dot{\theta} \implies \dot{\theta} = \frac{p_\theta}{m\ell^2}$$
$$p_\phi = \frac{\partial T}{\partial\dot{\phi}} = m\ell^2\sin^2\theta\dot{\phi} \implies \dot{\phi} = \frac{p_\phi}{m\ell^2\sin^2\theta}$$

So therefore, we can rewrite T:

$$T = \frac{1}{2m\ell^2}\left(p_\theta^2 + \frac{p_\phi^2}{\sin^2\theta}\right)$$

In the case where $p_\phi = 0$, then the second term in the parentheses goes to zero, so therefore we're left with:

$$\mathcal{H} = \frac{p_\theta^2}{2m\ell^2} + mg\cos\theta$$

Which is the exact equation for a plane pendulum. □

---

b) Combine the term that depends on p_ϕ with the ordinary potential energy term to define an effective potential $V(\theta, p_\phi)$. Sketch V as a function of θ for several values of p_ϕ, making sure to include $p_\phi = 0$.

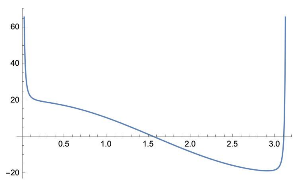
Solution: Rewirting the Hamiltonian a bit:

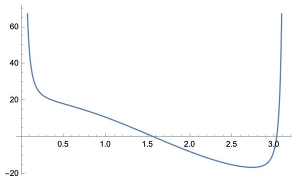
$$\mathcal{H} = \frac{p_\theta^2}{2m\ell^2} + \frac{p_\phi^2}{2m\ell^2\sin^2\theta} + mg\cos\theta$$

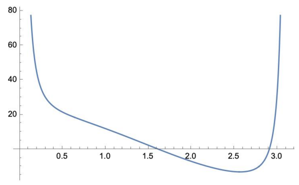
Then it's clear that if we want only the p_θ term to remain, then we would want the effective potential to be

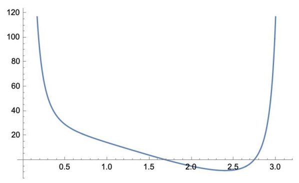
$$V(\theta, p_\phi) = \frac{p_\phi^2}{2m\ell^2\sin^2\theta} + mg\cos\theta$$

The sketches for $V(\theta, p_\phi)$ for $\theta \in [0, \pi]$ are shown below, first for $p_\phi = 1, 3, 6, 10$ in that order, from left to right:

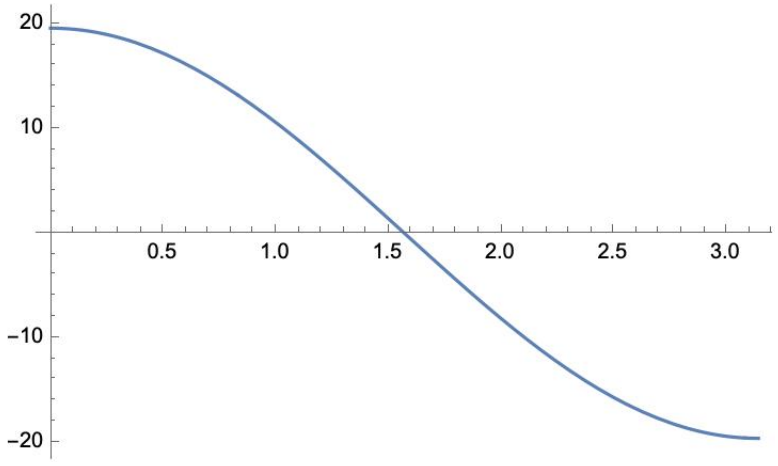








For $p_\phi = 0$, this is what we get over the same interval $\theta \in [0, \pi]$:



$\square$

c) Discuss the features of the motion, pointing out the differences between $p_\phi = 0$ and $p_\phi \neq 0$. Discuss the limiting case of the conical pendulum, $\theta = C$, a constant, with reference to the $V - \theta$ diagram.

Solution: Firstly, the motion looks very different for $p_\phi = 0$ as opposed to $p_\phi \neq 0$. Perhaps the most noticeable difference is the fact that with the case where $p_\phi \neq 0$, we get a global minimum at a $\theta \in (0, \pi)$, whose specific value is determined by p_ϕ itself. In the case where $p_\phi = 0$, we get a global minimum at $\theta = \pi$ – this makes sense, since there is no angular component to "carry" the mass around the sphere.

In terms of the motion itself: when $p_\phi \neq 0$, then we see that this leads to $\theta \neq 0$, implying that the mass travels around the sphere. For larger values of p_ϕ, we can see that the local minimum of the curve approaches $\frac{\pi}{2}$, this corresponds to the point when the mass rotates around the axis horizontally.

To analyze the conical pendulum, it's useful to first notice that $\frac{\partial \mathcal{H}}{\partial \phi} = 0$, so therefore p_ϕ is a constant. Because of this, it means that the pendulum will circulate the vertical axis about which the pendulum is rotating with both a constant polar angle and constant azimuthal angular velocity In other words, $\dot{\phi}$ is a constant throughout the motion. $\square$

Problem Set 3 problems

Question 1: Double pendulum

Learning objectives

In this question you will:

- study the double pendulum theoretically using Lagrangian mechanics
- numerically solve differential equations that can't be solved analytically
- assess the effectiveness of numerics by comparing theoretical expectations to numerical results
- understand what people mean by "chaos" by analysing a chaotic system

The double pendulum is pictured below:

1a.

Show that the Lagrangian of the system is given by

$$L = \tfrac{1}{2}(m_1 + m_2)L_1^2\dot{\phi}_1^2 + \tfrac{1}{2}m_2L_2^2\dot{\phi}_2^2 + m_2L_1L_2\dot{\phi}_1\dot{\phi}_2\cos(\phi_2 - \phi_1) - (m_1 + m_2)gL_1(1 -$$
$$- m_2gL_2(1 - \cos\phi_2).$$

Write your answer here

1b.

Find the equations of motion.

Write your answer here

1c.

Under the small angle approximation $\phi_1, \phi_2 \ll 1$ (keeping only first-order terms), find the normal modes of oscillation, which is pictured below for a simple choice of parameters.

The normal modes for $m_1 = m_1$, $L_1 = L_2$, and time measured in natural units. (Angles are exaggerated; this is only valid in the small-angle regime.)

Write your answer here

1d.

Now use your theoretical solution for small angles to fill in the following Python function that, given the initial state of the double pendulum, returns the states at some specified times. Use the convention $(\phi_1, \phi_2, \dot{\phi}_1, \dot{\phi}_2)^T$ for the state. (Hint: transform to the basis of normal modes and then transform back. The usual mapping of $\mathbb{R}^{2n}$ to $\mathbb{C}^n$ might be useful here.)

In [3]:
```python
import numpy as np

def small_angle_soln(initial_state, tmax, N, m1=1, m2=1, L1=1, L2=1, g=1):
    """
    Returns: tuple (times, states) where:
    times is an array of shape (N,) evenly sampled from 0 to tmax,
    states is array of shape (4,N) containing corresponding states in
        the format (phi_1, phi_2, d/dt phi_1, d/dt phi_2)
    """
    times = np.linspace(0,tmax,N)
    phi_1 = initial_state[0]
    phi_2 = initial_state[1]
    phidot1 = initial_state[2]
    phidot2 = initial_state[3]

    omega1 = np.sqrt((g*(L1 + L2) * (m1 + m2) + np.sqrt(g**2 * (L1 + L2)**2
    omega2 = np.sqrt((g*(L1 + L2) * (m1 + m2) - np.sqrt(g**2 * (L1 + L2)**2

    a1 = L1* omega1**2/ (L2 * omega1**2 - g)
    a2 = L1* omega2**2/ (L2 * omega2**2 - g)

    c1cos = (a2 * phi_1 - phi_2)/(a2 - a1) # c1 cos delta2
    c2cos = (a1 * phi_1 - phi_2)/(a1 - a2) # c1 cos delta2

    c1sin = -(a2 * phidot1 - phidot2)/(omega1 *(a2 - a1))
    c2sin = -(a1 * phidot1 - phidot2)/(omega2 *(a1 - a2))

    c1 = np.sqrt(c1cos**2 + c1sin**2)
    c2 = np.sqrt(c2cos**2 + c2sin**2)

    delta1 = np.arctan2(c1sin,c1cos)
    delta2 = np.arctan2(c2sin,c2cos)

    phi1 = c1 * np.cos(omega1 * times + delta1) + c2 * np.cos(omega2 * times
    phi2 = a1 * c1 * np.cos(omega1 * times + delta1) + a2 * c2 * np.cos(omeg
    dotphi1 = -omega1 * c1 * np.sin(omega1*times + delta1) + -omega2 * c2 *r
    dotphi2 = -omega1 * a1 * c1 * np.sin(omega1*times + delta1) + -omega2 *

    return times, np.array([phi1, phi2, dotphi1, dotphi2])
```

1e.

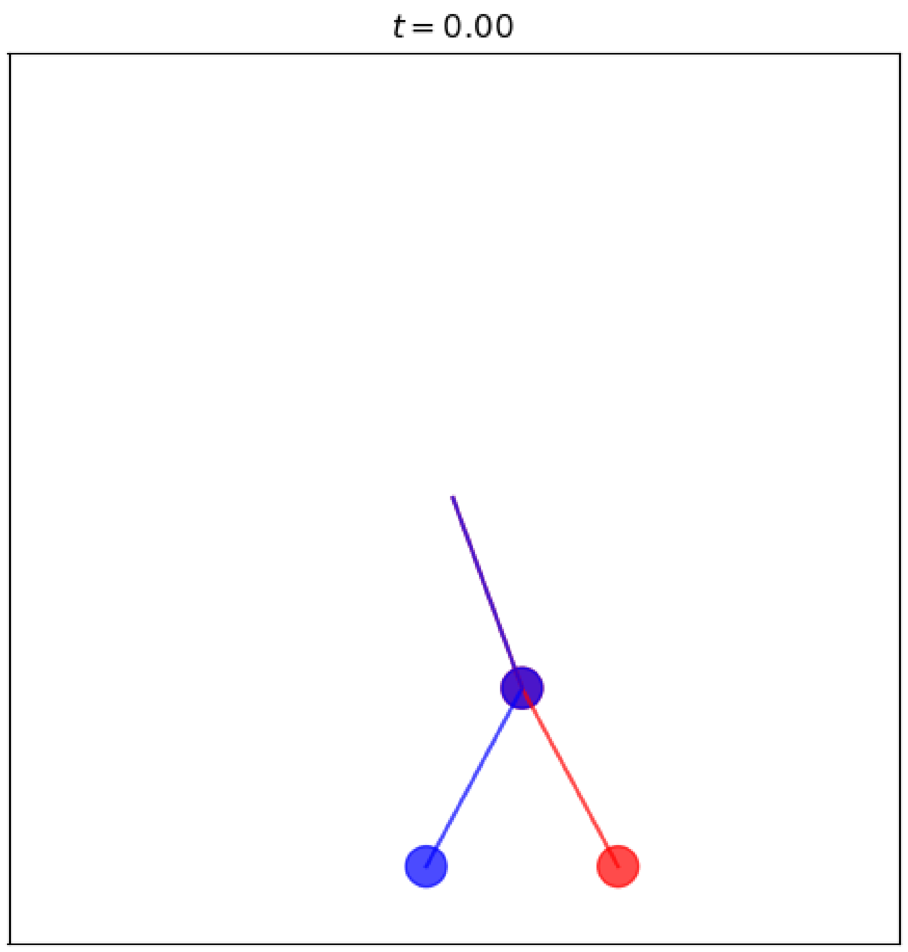
The following cells contains a visualisation function that animates the double pendulum given a list of times and state vectors. You might find it useful.

```
In [4]: %matplotlib inline
        from matplotlib import pyplot as plt
        from matplotlib.animation import FuncAnimation
        from IPython.display import display,HTML

        plt.rcParams["animation.embed_limit"] = 200
        #sets the max animation size to 200MB so that you can make long animations i

        def animate(times, states, c="k", m1=1, m2=1, L1=1, L2=1, g=1, labels=None):
            """
            Returns animation of pendula (or pendulum).
            times must be array of shape (N,).
            states must be array of shape (4,N), or list-like of n such arrays. (n i
                states must follow the usual convention.
            c must be list-like of length n. c[i] will be the colour of pendulum i.
            m1...g are problem parameters (g has no effect here)
            if labels are provided, must be list-like of length n. labels[i] will be
            Returns nothing, but displays the animation.
            """
            #change matplotlib backend so plots aren't displayed (only animation is)
            %matplotlib agg
            n = len(c) #number of pendula
            N = len(times) #number of frames
            xs, ys = [], [] #positions of pendula (store them to update them during
            strings = [] #line objects depicting strings (store them to update them
            pendula = [] #(tuples of) circle objects depiciting masses (store them t
            f = plt.figure(figsize=(6,6))
            labels = [None]*n if labels is None else labels
            for i in range(n):
                state = states[i] if n>1 else angles #get (4,N) state vector array
                x,y = np.zeros((N,3)), np.zeros((N,3)) #positions of string ends (wh
                y[:,1] = -L1*np.cos(state[0])
                x[:,1] = L1*np.sin(state[0])
                y[:,2] = y[:,1] - L2*np.cos(state[1])
                x[:,2] = x[:,1] + L2*np.sin(state[1])
                xs.append(x) #store positions
                ys.append(y)
                args = {"color": c[i], "alpha": np.sqrt(1/n)} #plotting args
                strings.append(plt.plot(x[0], y[0], label=labels[i], **args)[0]) #pl
                pend = plt.Circle((x[0,1],y[0,1]), m1**(1/3)/10, **args), plt.Circle
                [plt.gca().add_artist(p) for p in pend] #plot masses at initial posn
                pendula.append(pend) #store circle
            tit = plt.title(f"$t=0$") #store title object to update time
            padding = m2**(1/3)/10+0.1
            plt.xlim(-L1-L2-padding,L1+L2+padding) #set bounds so pundulum is always
            plt.ylim(-L1-L2-padding,L1+L2+padding)
            to_ret = [tit]+strings #list of things to be updated at each frame (for
            for p in pendula:
                to_ret += list(p)
            plt.xticks([]) #don't display the scale of axes
            plt.yticks([])
            plt.gca().set_aspect("equal") #set aspect ratio to 1
            if labels[0] is not None: #add legend if wanted
                plt.legend(frameon=False, loc="upper left")

            def update(j):
```

```
        tit.set_text(f"$t={times[j]:.2f}$") #update time in title
        for i in range(n):
            strings[i].set_data(xs[i][j],ys[i][j]) #update string ends
            pendula[i][0].set_center((xs[i][j,1],ys[i][j,1])) #update first
            pendula[i][1].set_center((xs[i][j,2],ys[i][j,2])) #update seconc
        return to_ret #return changed elements for blitting

    anim = FuncAnimation(f, update, range(N), interval=1e3/24, blit=True) #2
    display(HTML(anim.to_jshtml())) #output animation
    %matplotlib inline
```

As a test of your `small_angle_soln()`, and to show you how to use the `animate()` function, fill in the appropriate initial states to recover the normal modes for $m_1 = m_2$ and $L_1 = L_2$, and compare your animation to the one embedded above.

In [5]:
```
initial_state_1 = np.deg2rad([20, np.sqrt(2)* 20, 1.5, np.sqrt(2) * 1.5])
initial_state_2 = np.deg2rad([20, -(np.sqrt(2))*20, 1.5, -(np.sqrt(2)) * 1.5

tmax = 20
N = 200

times, normal_mode_1 = small_angle_soln(initial_state_1, tmax, N)
times, normal_mode_2 = small_angle_soln(initial_state_2, tmax, N)


animate(times, (normal_mode_1, normal_mode_2), "rb")
```



Once [✓ Loop] Reflect

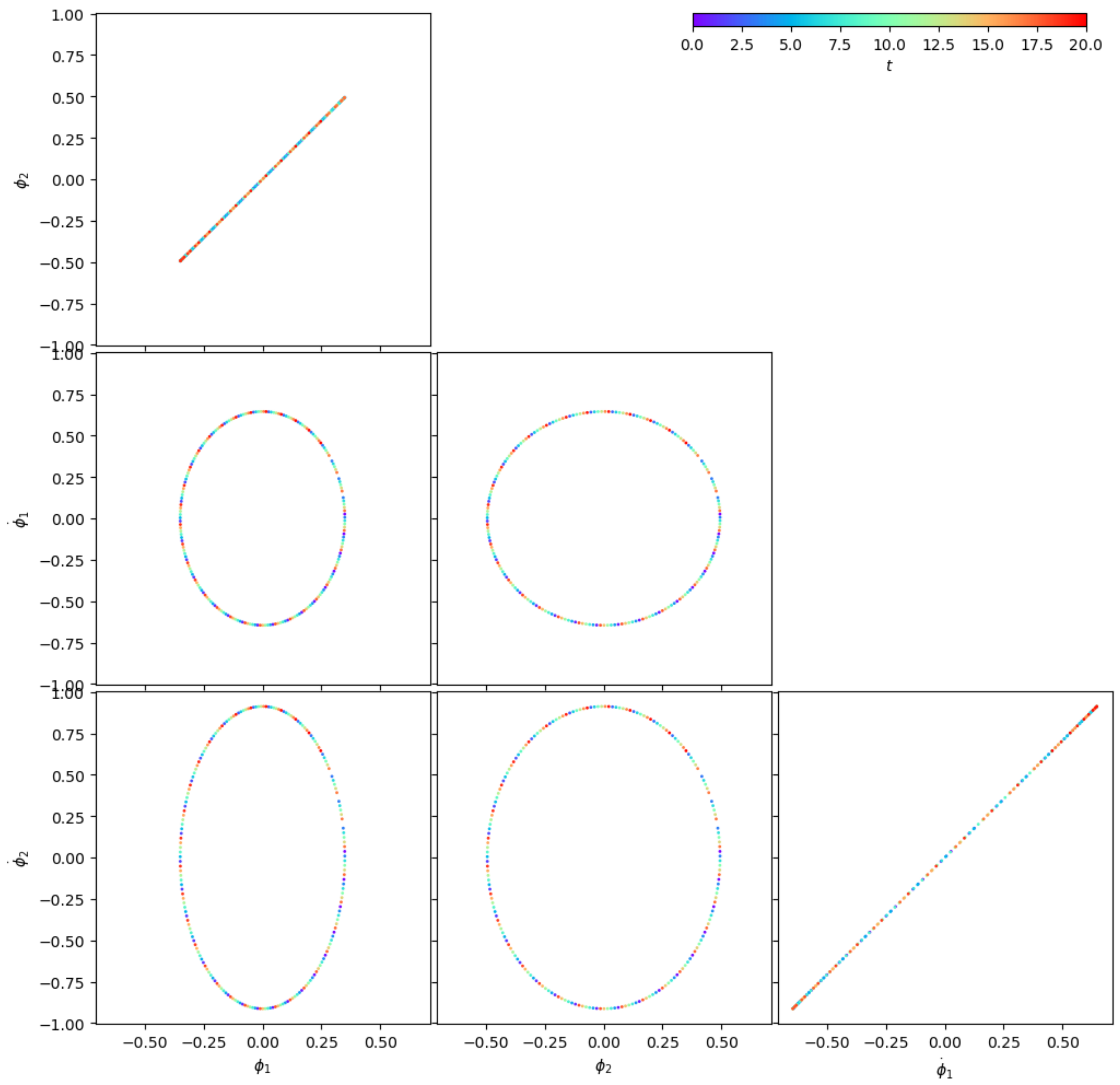
1f.

The following cells contain a visualisation function that plots a sampled trajectory in phase space.

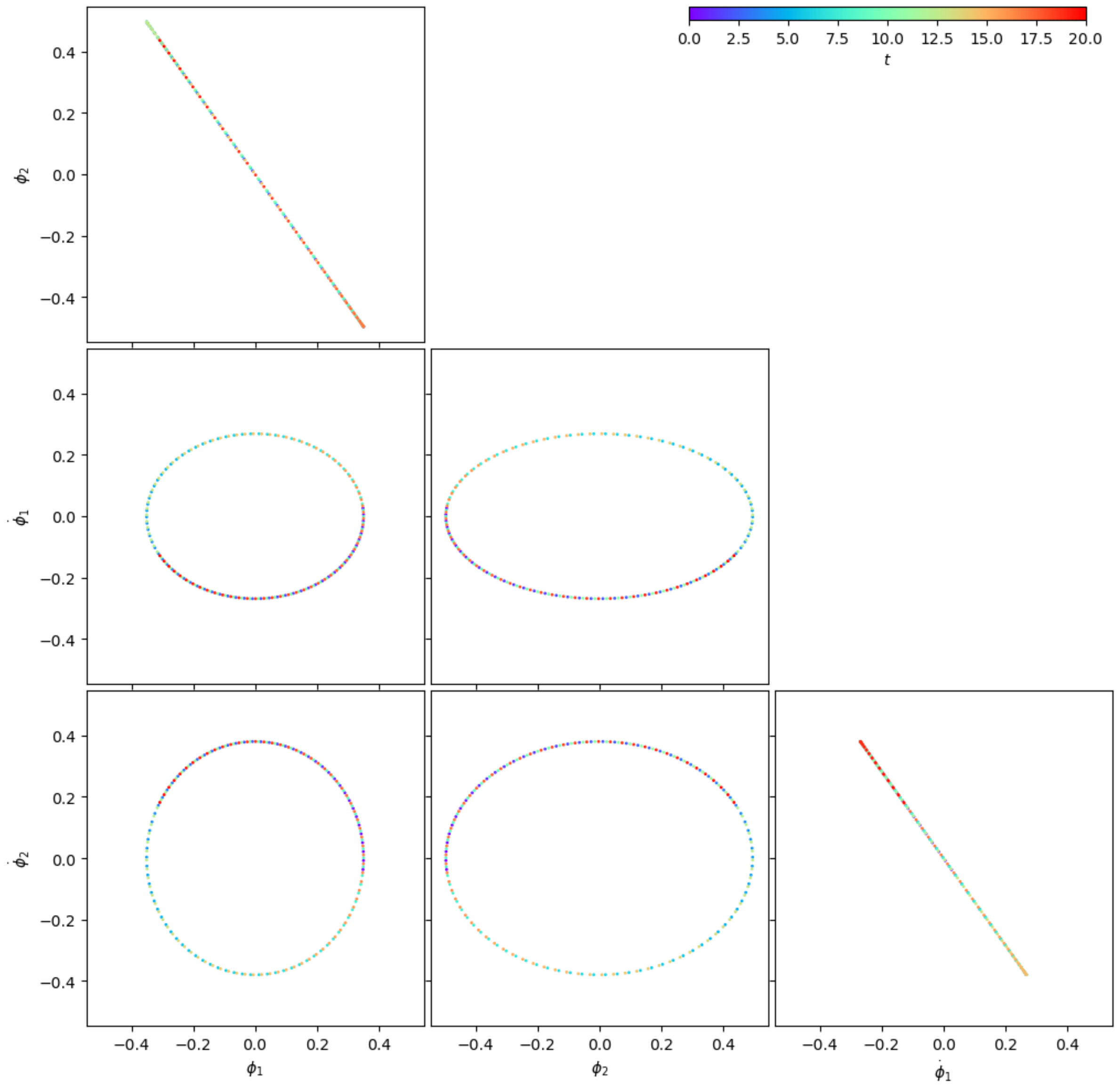
In [6]:
```python
from matplotlib.colors import Normalize
from matplotlib.colorbar import ColorbarBase
from matplotlib import cm as cmaps

def plot_phase_space(times, states, size=1, cmap=cmaps.rainbow):
    """
    times must be array of shape (N,) and states must be array of shape (4,N
    size is the size of marker (you might want to reduce it if N is large).
    cmap must be a matplotlib colourmap.
    Returns nothing, plots the sampled trajectory in 4-d phase space as 6 2-
    """
    f,ax = plt.subplots(3, 3, sharex=True, sharey=True, figsize=(12,12), gri
    labels = [r"$\phi_1$", r"$\phi_2$", r"$\dot\phi_1$", r"$\dot\phi_2$"]
    args = {"c": times, "s": size, "cmap": cmap}
    for i in range(3):
        ax[i,0].set_ylabel(labels[i+1])
        ax[-1,i].set_xlabel(labels[i])
        for j in range(i+1):
            ax[i,j].scatter(states[j], states[i+1], **args)
        for j in range(i+1,3):
            ax[i,j].axis("off")
    cbax =  f.add_axes([0.58,0.87,0.3,.01])
    ColorbarBase(cbax, cmap=cmap, norm=Normalize(times[0], times[-1]), orier
```

Run the following cell to visualise the solutions you animated above in phase space. Is this what you expect?

In [7]:
```
plot_phase_space(times, normal_mode_1)
```



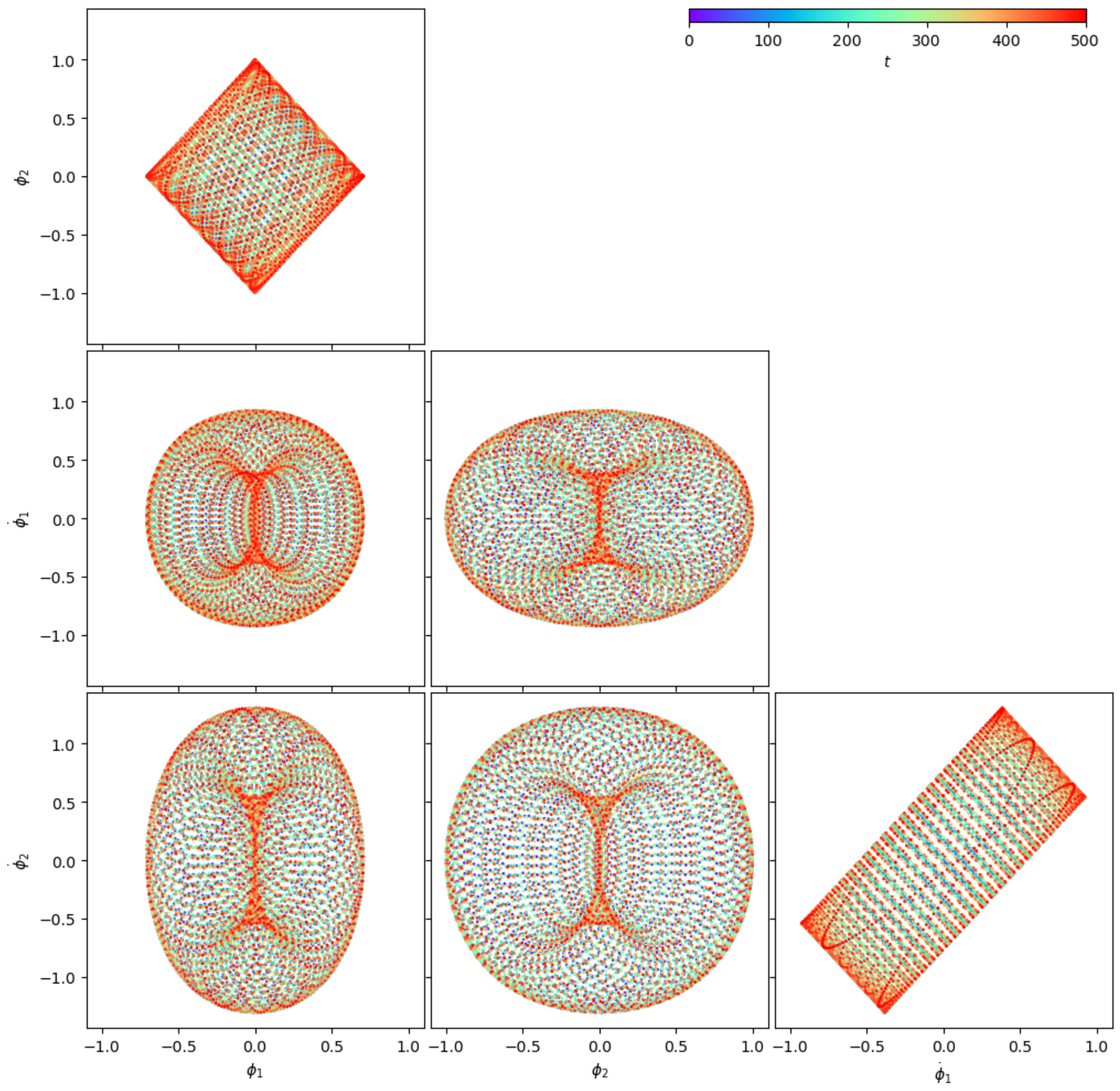
```
In [8]: plot_phase_space(times, normal_mode_2)
```



Now play around with the system. Plot phase space trajectories for some other initial states that are not normal modes. Use large values for $t_{\max}$ and N. What does the phase space trajectory look like? Is it periodic, or dense in some allowed phase space?

An *ergodic* system is a system in which from any starting state, one reaches arbitrarily close to any given state at some time. Does this system (linearised under the small-angle approximation) display ergodicity?

```
In [9]: plot_phase_space(*small_angle_soln([0,1,0,0],500,10000))
```



In general, the system appears to be ergodic over a visible area. Also, the region over which the points exist are constrained by initial conditions, which set limitations on parametrs such as the total energy within the system.

When the motion is precisely a normal mode, then we see that the area in phase space is very small, almost nonexistent. This makes sense intuitively, since we expect that the normal modes are oscillatory in nature, and do not diverge away from the initial solution. Hence, their plot in phase space shouldn't cover an area. We do not expect this behavior for a system that is a linear combination of the normal modes (with nonzero contributions from both modes), and in this case we see that the motion fills up a defined area.

1g.

Rewrite the (full) equations of motion as a (non-linear) four-dimensional first-order ODE. Use the same convention for the state, $(\phi_1, \phi_2, \dot\phi_1, \dot\phi_2)^T$.

From the previous homework, we know that the equations of motion are:

$$\begin{aligned} m_2 L_1 \dot\phi_1 \dot\phi_2 \sin(\phi_2 - \phi_1) - (m_1 + m_2) g L_1 \sin\phi_1 = (m_1 + m_2) L_1^2 \ddot\phi_1 \\ + m_2 L_1 L_2 \left[\ddot\phi_2 \cos(\phi_2 - \phi_1) - \dot\phi_2 \sin(\phi_2 - \phi_1)(\dot\phi_2 - \dot\phi_1)\right] \end{aligned}$$

And similarly,

$$\begin{aligned} -m_2 L_1 L_2 \dot\phi_1 \dot\phi_2 \sin(\phi_2 - \phi_1) - m_2 g L_2 \sin\phi_2 = m_2 L_2^2 \ddot\phi_2 \\ + m_2 L_1 L_2 \left[\ddot\phi_! \cos(\phi_2 - \phi_1) - \dot\phi_1 \sin(\phi_2 - \phi_1)(\dot\phi_2 - \dot\phi_1)\right] \end{aligned}$$

We want to rewrite this into the form

$$A\ddot\phi_1 + B\ddot\phi_2 = C$$

Therefore, it's natural to write:

$$\begin{aligned} A_1 &= (m_1 + m_2) L_1^2 \\ A_2 = B_1 &= m_2 L_1 L_2 \cos(\phi_2 - \phi_1) \\ B_2 &= m_2 L_2^2 \\ C_1 &= m_2 L_1 L_2 \dot\phi_2^2 \sin(\phi_2 - \phi_1) - (m_1 + m_2) g L_1 \sin\phi_1 \\ C_2 &= -m_2 L_1 L_2 \dot\phi_1^2 \sin(\phi_2 - \phi_1) - m_2 g L_2 \sin\phi_2 \end{aligned}$$

We can write this as a matrix equation:

$$\begin{pmatrix} A_1 & B_1 \\ B_1 & B_2 \end{pmatrix} \begin{pmatrix} \ddot\phi_1 \\ \ddot\phi_2 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}$$

Solving this for $\ddot\phi_1$ and ϕ_2, we get:

$$\begin{aligned} \ddot\phi_1 &= \frac{B_2 C_1 - B_1 C_2}{A_1 B_2 - B_1^2} \\ \ddot\phi_2 &= \frac{-B_1 C_1 + A_1 C_2}{A_1 B_2 - B_1^2} \end{aligned}$$

1h.

Implement your answer above in the following function:

```
In [54]: def derivative(state, m1=1, m2=1, L1=1, L2=1, g=1):
             """Given state of shape (4,) as well as the problem parameters,
             returns an array of shape (4,) representing the time-derivative of the s

             A1 = (m1 + m2)* L1**2
             B1 = m2 * L1 * L2 * np.cos(state[1] - state[0])
             B2 = m2 * L2**2
```

```
    C1 = m2 * L1 * L2 * state[3]**2 * np.sin(state[1] - state[0]) - (m1 + m2
    C2 = -m2 * L1 * L2 * state[2]**2 * np.sin(state[1] - state[0]) - m2*g*L2
    der = np.array([state[2], state[3], (B2 * C1 - B1 * C2)/(A1*B2 - B1**2),
    return der
```

1i.

Solve the EoMs analytically.

Lol jk, let's use some differential equation solvers to solve the full, non-linear EoMs which we can't solve analytically. The functions given below are implementations of the Euler, Symplectic Euler, and a 5th order Runge-Kutta method (these methods are introduced in the introductory Python notebooks) and follow the same conventions (and have the same call signature) as `small_angle_soln()`. They will solve the ODE defined by your implementation of `derivative()`, which, if you did everything right, are the EoMs of the double pendulum.

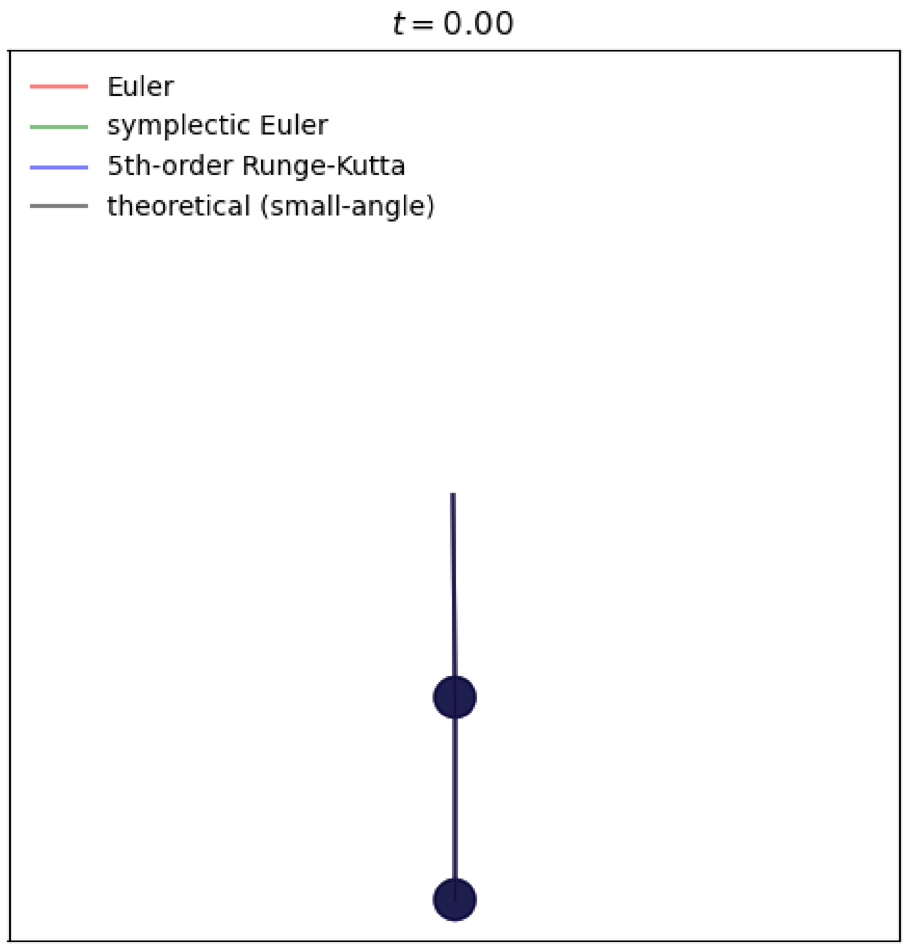
In [55]:
```
from scipy.integrate import solve_ivp

def euler(initial_state, tmax, N, **extra_args):
    times = np.linspace(0,tmax,N)
    dt = times[1]-times[0]
    states = np.zeros((N,4))
    states[0] = initial_state
    for i in range(N-1):
        states[i+1] = states[i] + derivative(states[i], **extra_args)*dt
    return times, states.T

def symplectic_euler(initial_state, tmax, N, **extra_args):
    times = np.linspace(0,tmax,N)
    dt = times[1]-times[0]
    states = np.zeros((N,4))
    states[0] = initial_state
    for i in range(N-1):
        states[i+1,2:] = states[i,2:] + derivative(states[i], **extra_args)[
        states[i+1,:2] = states[i,:2] + states[i+1,2:]*dt
    return times, states.T

def runge_kutta(initial_state, tmax, N, **extra_args):
    times = np.linspace(0,tmax,N)
    fn = lambda t,y: derivative(y, **extra_args)
    soln = solve_ivp(fn, (0, tmax), initial_state, max_step=tmax/(N+1), dens
    return times, soln(times)
```

Here is a function that will plot phase-space plots (and, optionally, an animation) of trajectories sampled using each of the three methods (and, optionally, the theoretical small-angle solution) for comparison. There is also a helper function to bring angles within the interval $[-\pi, \pi)$, which might be useful later.

In [56]:
```
def bound_angles(states):
    """Returns copy of states, where angles are in the interval [-pi,pi)"""
    states = states.copy()
    states[:2] = (states[:2]+np.pi)%(2*np.pi)-np.pi
    return states

def compare_methods(*args, theo=False, anim=True, energies=False, **extra_ar
    """
    First three args must be initial_state, tmax and N.
    theo is boolean for whether or not to include theoretical small-angle so
    anim is boolean for whether or not to animate the trajectories (takes a
    energies is boolean for whether or not to plot the energies as well. (re
    extra_args are problem parameters.
    Returns nothing, solves ODE using the 3-4 methods, and plots phase-space
    """
    labels = ["Euler", "symplectic Euler", "5th-order Runge-Kutta"]
    methods = [euler, symplectic_euler, runge_kutta]
    colours = "rgb"
    if theo:
        labels += ["theoretical (small-angle)"]
        methods += [small_angle_soln]
        colours ="rgbk"
    ys=[]
    for meth in methods:
        t,a = meth(*args, **extra_args)
        ys.append(a)
    if anim:
        extra_args["labels"] = labels
        animate(t, ys, colours, **extra_args)
    for i in range(len(labels)):
        plot_phase_space(t,bound_angles(ys[i]))
        plt.suptitle(labels[i])
    if energies:
        plt.figure()
        for i in range(len(labels)):
            plt.plot(t,energy(ys[i],**extra_args),label=labels[i])
        plt.axhline(energy(args[0],**extra_args),color="k",ls="--",label="tr
        plt.legend(frameon=False)
        plt.xlabel("time")
        plt.ylabel("energy")
        plt.yscale("log")
```

For a small angle, compare the three methods to the theoretical solution. Which methods are able to capture the continuous behaviour, and which aren't? Simply running the following cell will give you lots of information.

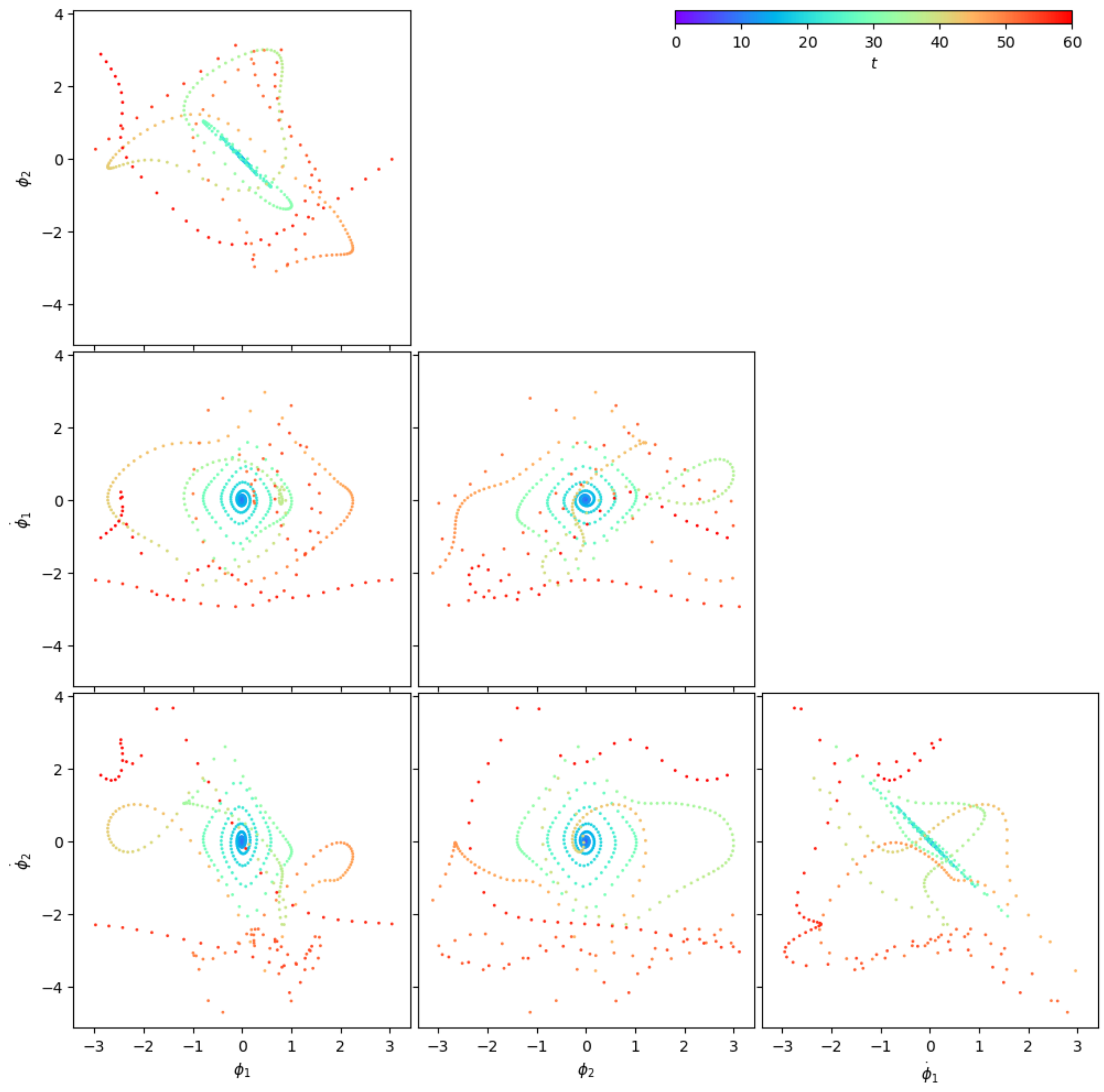
```
In [58]: %time compare_methods([0.01,0,0,0], 60, 500, theo=True)
```



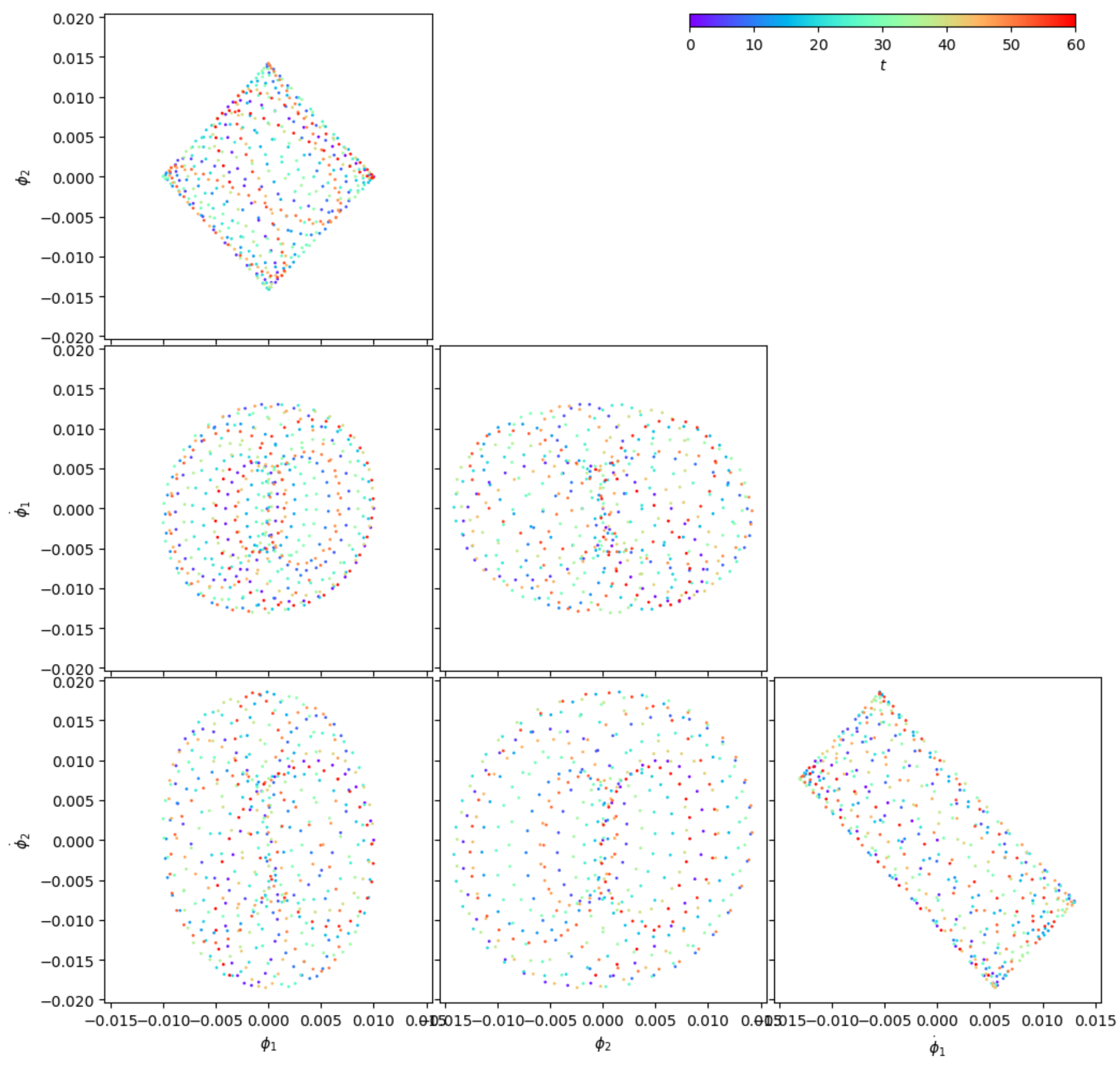
( ) Once [✓ Loop] ( ) Reflect

```
CPU times: user 25.7 s, sys: 564 ms, total: 26.3 s
Wall time: 26.3 s
```

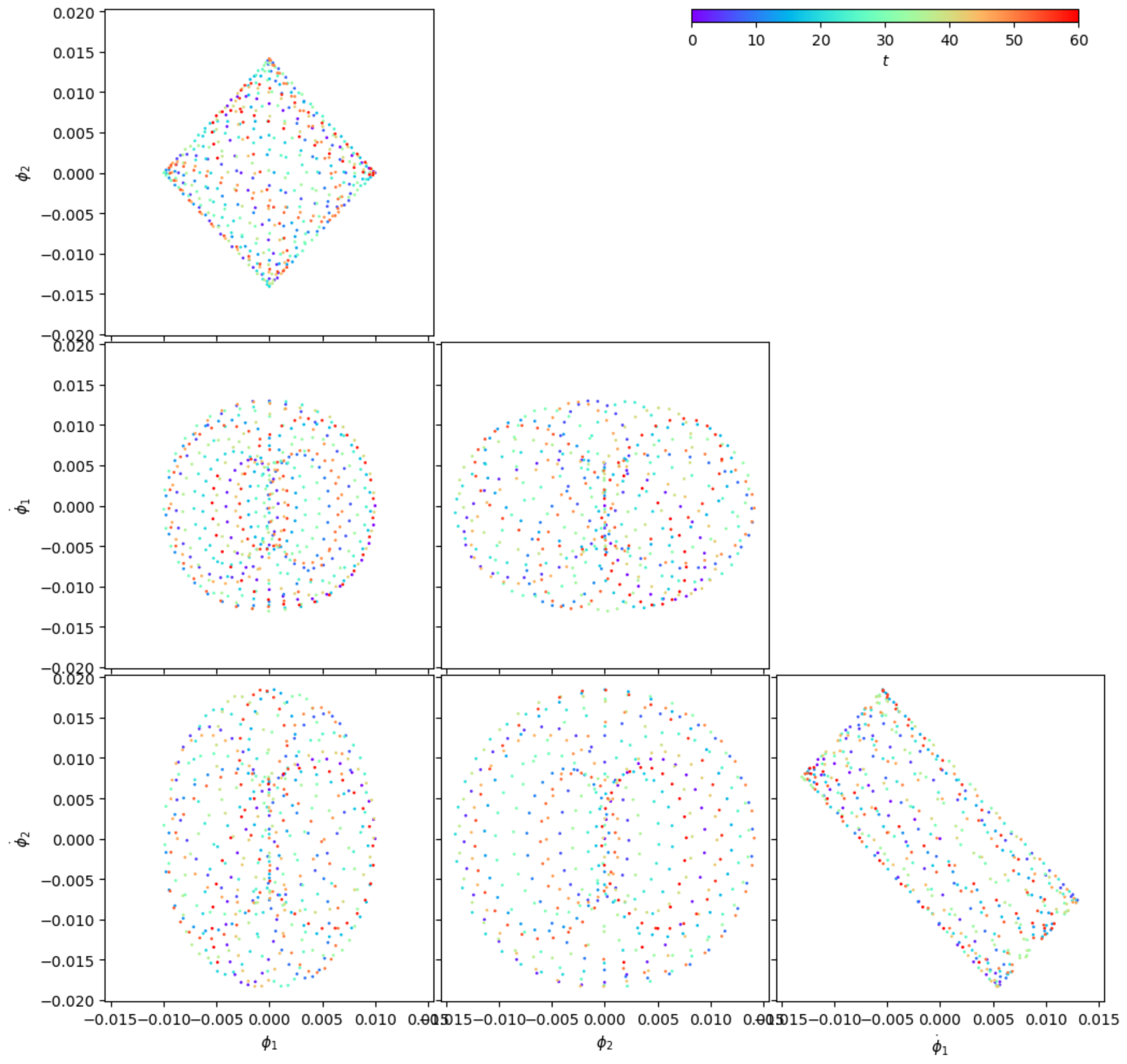
Euler



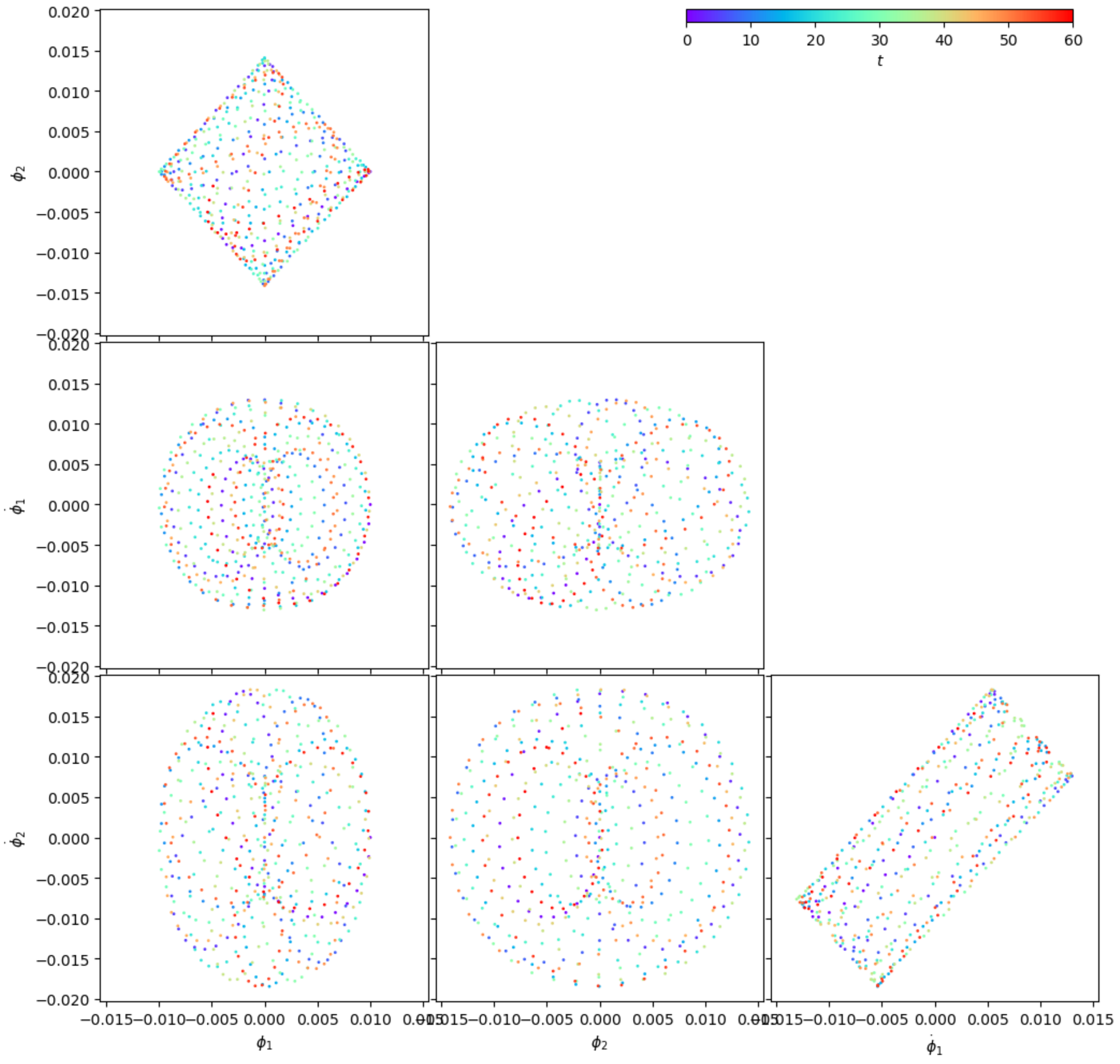
symplectic Euler



5th-order Runge-Kutta



theoretical (small-angle)



It seems that the Euler method loses track of the system over time, and diverges away from the actual motion the fastest. The symplectic Euler and Runge-Kutta methods both seem to do equally well when compared against one another, but its calculated motion is different than that of the theoretical small angle solution. This is espcially clear in the $\dot{\phi}_1$ vs. $\dot{\phi}_2$ plot, where the rectangle that bounds the points is positively correlated in the theoretical solution but negative in the simulated ones. This is interesting and I can't really come up with a good reason for it.

1j.

Compare the three methods outside of the small-angle regime. For how long do they display qualitatively similar behaviour? Which method, if any, can we trust?

```
In [41]: #Write your answer here
```

```
0.0009999998333333417
```

1k.

A good test that the differential equation solvers aren't accumulating errors is to check if constants of the motion are conserved (the ODE solvers don't know about conserved quantities, so they don't enforce them; if they did they would be more efficient). Here, the only (known) conserved quantity is energy. Why is that?

Write your answer here

1l.

Implement the following function to calculate the energies of state vectors. After implementing this function, you can pass `energies=True` to `compare_methods()` to also plot the energies of the trajectories, to check if it is constant.

In [15]:
```python
def energy(states, m1=1, m2=1, L1=1, L2=1, g=1):
    """states is array of shape (4,N). Returns energies in array of shape (N
    return #fill me in
```

1m.

Now, integrating to large times (you might want to turn off animation), compare the three methods outside of the small-angle regime, also comparing the energies. How much can we trust the best method, and upto what sort of time scale?

In [16]:
```python
#Write your answer here
```

1n.

You might have heard that the double pendulum is chaotic. This means that it is extremely sensitive to initial conditions. To demonstrate this, make an animation of a few pendula that start off very close together but then diverge. Verify that the solver you use isn't giving you terribly unreasonable results.

In [17]:
```python
#Write your answer here
```

1o.

Chaotic systems aren't just unstable to perturbations: they are **extremely** unstable to perturbations. Perturbations must grow exponentially for a system to be chaotic. Choose an appropriate initial state and find its trajectory. Then perturb it slightly in a few different ways, and track the differences in the trajectories. How quickly do perturbations grow? Is the double pendulum chaotic inside the small-angle regime (remember that most trajectories are dense)? How about outside it?

```
In [18]: #Write your answer here
```

1p.

Suppose an experimenter initialises a pendulum to some state. Now, in the real world, we only have access to a finite level of accuracy. Assuming that the error in the initialisation/measurement of each parameter is independent and Gaussian, argue that the pendulum is not described by a single state, but by a Gaussian distribution over states. What happens to this distribution over time?

Write your answer here