Eric Du — Physics 105

HW 12 — Analytic Mechanics — April 29, 2023

Collaborators

I worked with **Adarsh Iyer, Aren Martinian** and **Andrew Binder** to complete this homework.

Problem 1

Calculate the solid angles subtended by the Moon and by the Sun, both as seen from the Earth.

Solution: From the textbook, we know that the solid angle is given by $\Delta\Omega = \frac{A}{r^2}$, so all we need to do is find the cross sectional area of each celestial body (given by πR^2), and divide that by the mean distance between the Earth and the Sun/Moon. Plugging in numbers for the Sun:

$$\Delta\Omega_{\text{sun}} = 6.79 \times 10^{-5} \text{ sr}$$

And then for the moon:

$$\Delta\Omega_{\text{moon}} = 6.4 \times 10^{-5} \text{ sr}$$

This actually makes sense, since we see that the sun and moon look approximately the same size, despite the sun being much bigger, because the Sun is also much farther away than the moon. □

Problem 2

One can set up a two-dimensional scattering theory, which could be applied to puck projectiles sliding on an ice rink and colliding with various target obstacles. The cross section would be the effective width of a target, and the differential cross section $d\sigma/d\theta$ would give the number of projectiles scattered in the angle $d\theta$.

a) Show that the two-dimensional analog of Eq. (14.23) is $d\sigma/d\theta = |db/d\theta|$. (Note that in two-dimensional scattering it is convenient to take $\theta \in [-\pi, \pi]$.

Solution: The small patch of particles that scatter into the solid angle $d\theta$ is now given by a small patch of vertical height db, so therefore $d\sigma = db$. Further, since the system is symmetric for $\pm\theta$, then we can combine this information and solve for the absolute value instead. Therefore:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\theta} = \left|\frac{\mathrm{d}b}{\mathrm{d}\theta}\right|$$

□

---

b) Now consider the scattering of a small projectile off a hard "sphere" (actually a hard disk) of radius R pinned down to the ice. Find the differential cross section.

Solution: We follow a very similar argument to that of the hard ball scattering. Consider a particle with an impact parameter b. Due to the geometry of the problem, we see that $b = R\sin\alpha$. Further, due to momentum conservation, this scattering must obey the law of reflection, which implies that $\theta = \pi - 2\alpha$ (see diagram). Therefore, we can write

$$b = R\left|\sin\left(\frac{\pi-\theta}{2}\right)\right|$$

Now, taking the derivative with respect to θ, we get:

$$\left|\frac{\mathrm{d}b}{\mathrm{d}\theta}\right| = \left|\frac{R}{2}\cos\left(\frac{\pi-\theta}{2}\right)\right|$$

□

---

c) By integrating your answer to part (b), show that the total cross section is $2R$ as expected.

Solution: Here, we integrate with respect to θ, from the bounds $-\pi$ to π:

$$b = \frac{R}{2}\int_{-\pi}^{\pi}\left|\cos\left(\frac{\pi-\theta}{2}\right)\right| d\theta = \frac{R}{2}(4) = 2R$$

as desired. □

Problem 3

One of the first observations that suggested his nuclear model of the atom to Rutherford was that several alpha particles got scattered by metal foils into the backward hemisphere, $\pi/2 \leq \theta \leq \pi$ - an observation that was impossible to explain on the basis of rival atomic models, but emerged naturally from the nuclear model. In an early experiment, Geiger and Marsden measured the fraction of incident alphas scattered into the backward hemisphere off a platinum foil. By integrating the Rutherford cross section

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = \frac{\sigma_0(E)}{\sin^4(\theta/2)}, \text{ with } \sigma_0(E) = \frac{k^2q^2Q^2}{16E^2}$$

over the backward hemisphere, show that the cross section for scattering with $\theta \geq \pi/2$ should be $4\pi\sigma_0(E)$. Using the following numbers, predict the ratio $N_{sc}(\theta \geq \pi/2)/N_{in}$: thickness of platinum foil $\approx$ 3 pm, density = 21.4 gram/cm^3, atomic weight = 195, atomic number = 78, energy of incident alphas = 7.8 MeV. Compare your answer with their estimate that "of the incident α particles about 1 in 8000 was reflected". (that is, scattered into the backward hemisphere). Small as this fraction is, it was still far larger than any rival model of the atom could explain.

Solution: The integral we want to take is

$$\sigma = \int \frac{\sigma_0(E)}{\sin^4(\frac{\theta}{2})} d\Omega$$

Since $\sigma_0(E)$ is a constant, we can take it out of the integral. Therefore, we now have:

$$\sigma = \sigma_0(E) \int \frac{1}{\sin^4(\frac{\theta}{2})} d\Omega$$

We can do two things here: first, we rewrite $d\Omega = \sin\theta d\theta d\phi$. Then, we note our bounds of integration are $\theta \in [\pi/2, \pi]$, and $\phi \in [0, 2\pi]$. Since everything in the integral is independent of ϕ, we can just multiply by 2π, leaving us with:

$$\sigma = 2\pi\sigma_0(E) \int_{\frac{\pi}{2}}^{\pi} \frac{\sin\theta}{\sin^4(\frac{\theta}{2})} d\theta$$

Then, we can perform a u-substitution, letting $u = \frac{\theta}{2}$, so therefore $du = \frac{1}{2}d\theta$. This turns our integral into:

$$\begin{aligned}\sigma &= 2\pi\sigma_0(E) \int_{\frac{\pi}{4}}^{\frac{\pi}{2}} \frac{\sin 2u}{\sin^4 u}(2du) \\ &= 4\pi\sigma_0(E) \int_{\frac{\pi}{4}}^{\frac{\pi}{2}} \frac{2\cos u}{\sin^3 u} du \\ &= 8\pi\sigma_0(E) \int_{\frac{\pi}{4}}^{\frac{\pi}{2}} \frac{\cos u}{\sin^3 u} du\end{aligned}$$

From here, it's a simple u-substitution again: let $v = \sin u$, so $dv = \cos u du$. Therefore:

$$\begin{aligned}\sigma &= 8\pi\sigma_0(E) \int_{\frac{1}{\sqrt{2}}}^{1} \frac{dv}{v^3} \\ &= 8\pi\sigma_0(E)(\frac{1}{2}) \\ &= 4\pi\sigma_0(E)\end{aligned}$$

as desired. Plugging in numbers, we get:

$$\frac{N_{\text{sc}}}{N_{\text{in}}} = 1.292 \times 10^{-4}$$

this gives a frequency of about 1/8333, which is approximately what Rutherford got. □

Problem 4

The derivation of the Rutherford cross section was made simpler by the fortuitous cancellation of the factors of r in the integral, Eq. (14.30). Here is a method of finding the cross section which works, in principle, for any central force field: The general appearance of the scattering orbit is as shown in Figure 14.11. It is symmetric about the direction $\mathbf{u}$ of closest approach. If ψ is the projectile's polar angle, measured from the direction $\mathbf{u}$, then $\psi \to \pm\psi_0$ as $t \to \pm\infty$ and the scattering angles $\theta = \pi - 2\psi_0$, as in the following figure depicting nuclear scattering:

[insert tikz here]

The angle ψ_0 is equal to $\int_{t_0}^{\infty} \dot{\psi}(t)dt$ taken from the time of closest approach $t = t_0$ to $t = \infty$. We can rewrite this as

$$\int_{t_0}^{\infty} \frac{\dot{\psi}(t)}{\dot{r}(t)}\dot{r}(t)dt = \int_{r_0}^{\infty} \frac{\dot{\psi}(r)}{\dot{r}(r)}dr$$

where now r_0 is the distance of closest approach. Next rewrite ψ in terms of the angular momentum ℓ and r, and rewrite $\dot{r}$ in terms of the energy E and the effective potential U_{eff}. Having done all this you should be able to prove that

$$\theta = \pi - \frac{2}{b}\int_{r_0}^{\infty} \frac{(b/r)^2}{\sqrt{1-(b/r)^2 - U(r)/E}}$$

Provided this integral can be evaluated, it gives θ in terms of b, and hence the cross section. Fill in the details of this calculation to prove this formula.

Solution: First, we start by noticing that $\dot{\psi}$ is the polar angle, so $\dot{\psi}$ represents the angular velocity of the particle. Therefore, we can write the angular momentum as $\ell = mr^2\dot{\psi}$, which rearranges to

$$\dot{\psi} = \frac{\ell}{mr^2}$$

Next, we know that at large distance, we have $\ell = r \times p = rp\sin\theta$, and since $r\sin\theta = b$, then we have $l = p \cdot b$, and using the expression that $E = \frac{p^2}{2m}$ (conservation of energy), we get $\ell = b\sqrt{2mE}$, so therefore:

$$\dot{\psi} = \frac{\sqrt{2mE}b}{mr^2} = \sqrt{\frac{2E}{m}}\frac{b}{r^2}$$

Now, we have to handle $\dot{r}$. To do this, consider the conservation of energy

$$E = \frac{1}{2}m\dot{r}^2 + U_{\text{eff}}(r) = \frac{1}{2}m\dot{r}^2 + U(r) + \frac{\ell^2}{2mr^2}$$

The $\frac{\ell^2}{2mr^2}$ term can be simplified directly using our substitution of $\ell = \sqrt{2mE}b$, giving us

$$\frac{\ell^2}{2mr^2} = \frac{Eb^2}{r^2}$$

Next, solving for $\dot{r}$:

$$\dot{r} = \sqrt{\frac{2}{m}\left(E - U(r) - \frac{Eb^2}{r^2}\right)} = \sqrt{\frac{2E}{m}\left(1 - \frac{U}{E} - \frac{b^2}{r^2}\right)}$$

Therefore, putting it all together:

$$\frac{\dot{\psi}}{\dot{r}} = \frac{\sqrt{\frac{2E}{m}}\frac{b}{r^2}}{\sqrt{\frac{2E}{m}\left(1-\frac{U}{E}-\frac{b^2}{r^2}\right)}} = \frac{b/r^2}{\sqrt{1-\frac{U}{E}-\frac{b^2}{r^2}}}$$

Factoring out a b, we have:

$$\frac{1}{b}\frac{\dot{\psi}}{\dot{r}} = \frac{(b/r)^2}{\sqrt{1 - \frac{U}{E} - \frac{b^2}{rr}}}$$

Now we integrate from r_0 to ∞, so we get:

$$\theta = \pi - 2\int_{r_0}^{\infty} \frac{(b/r^2)dr}{\sqrt{1 - (b/r)^2 - U(r)/E}}$$

as desired. $\square$

Problem 5

Use the formula obtained in the previous problem to answer the following problems. You may use a computer to evaluate integrals.

a) Consider hard sphere scattering. What should you take to be $U(r)$? Find $b(\theta)$, and then $d\sigma/d\Omega$, and finally σ. Does your result for σ make sense?

Solution: For a hard sphere, we know that $U(r) = 0$ for $r > r_0$ (the radius of the sphere), and the location of closest approach is $r = r_0$, so therefore our integral for θ becomes:

$$\theta = \pi - \frac{2}{b}\int_{r_0}^{\infty} \frac{(b/r)^2}{\sqrt{1-(b/r)^2}} dr$$

we substitute $u = \frac{b}{r}$ so this integral becomes $\sin^{-1}(u)$, so then we get:

$$\theta = \pi - 2\sin^{-1}\left(\frac{b}{r_0}\right)$$

Therefore, we have:

$$b = r_0 \cos\left(\frac{\theta}{2}\right)$$

To find $\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}$, we then integrate this using

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = \frac{b}{\sin\theta}\left|\frac{\mathrm{d}b}{\mathrm{d}\theta}\right| = \frac{r_0^2}{4}$$

Now finally, we integrate this from $\theta \in [0,\pi]$ and $\phi \in [0, 2\pi]$:

$$\int \frac{r_0^2}{4} d\Omega = (4\pi)\frac{r_0^2}{4} = \pi r_0^2$$

this makes sense, since it's equal to the cross sectional area of the hard ball. □

---

b) Consider Rutherford scattering with $F = kqQ/r^2$. Find the values of $b(\theta)$, $d\sigma/d\Omega$ and σ.

Solution: Here, we need to solve:

$$\theta = \pi - \frac{2}{b}\int_{r_0}^{\infty} \frac{(b/r)^2}{\sqrt{1-\frac{b^2}{r^2}-\frac{kqQ}{rE}}}$$

First, we perform a u substitution of $u = \frac{b}{r}$, which simplifies the integral to:

$$\theta = \pi + 2\int_{\frac{b}{r_0}}^{0} \frac{u\ du}{\sqrt{1-u^2-\frac{kqQu}{bE}}}$$

We can then plug this into Mathematica, which gives us:

$$\theta = \pi - 2\tan^{-1}\left(\frac{2bE}{kqQ}\right)$$

Rearranging for b, we have:

$$b = \frac{kqQ}{2e}\tan\left(\frac{\pi-\theta}{2}\right) = \frac{kqQ}{2e}\cot\left(\frac{\theta}{2}\right)$$

Now we can get $\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}$:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = \frac{b}{\sin\theta}\left|\frac{\mathrm{d}b}{\mathrm{d}\theta}\right| = -\frac{b}{\sin\theta}\frac{kqQ}{4E}\csc^2\left(\frac{\theta}{2}\right)$$

Finally, integrating this over $d\Omega$, we get:

$$\sigma = -\frac{kqQb}{4E}\int \frac{\csc^2\left(\frac{\theta}{2}\right)}{\sin\theta}d\Omega = -\frac{kqQb}{4E}(2\pi)\int_0^\pi \frac{\csc^2\left(\frac{\theta}{2}\right)}{\sin\theta}d\theta$$

which does not converge. □

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

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

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 [1]:
```
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.

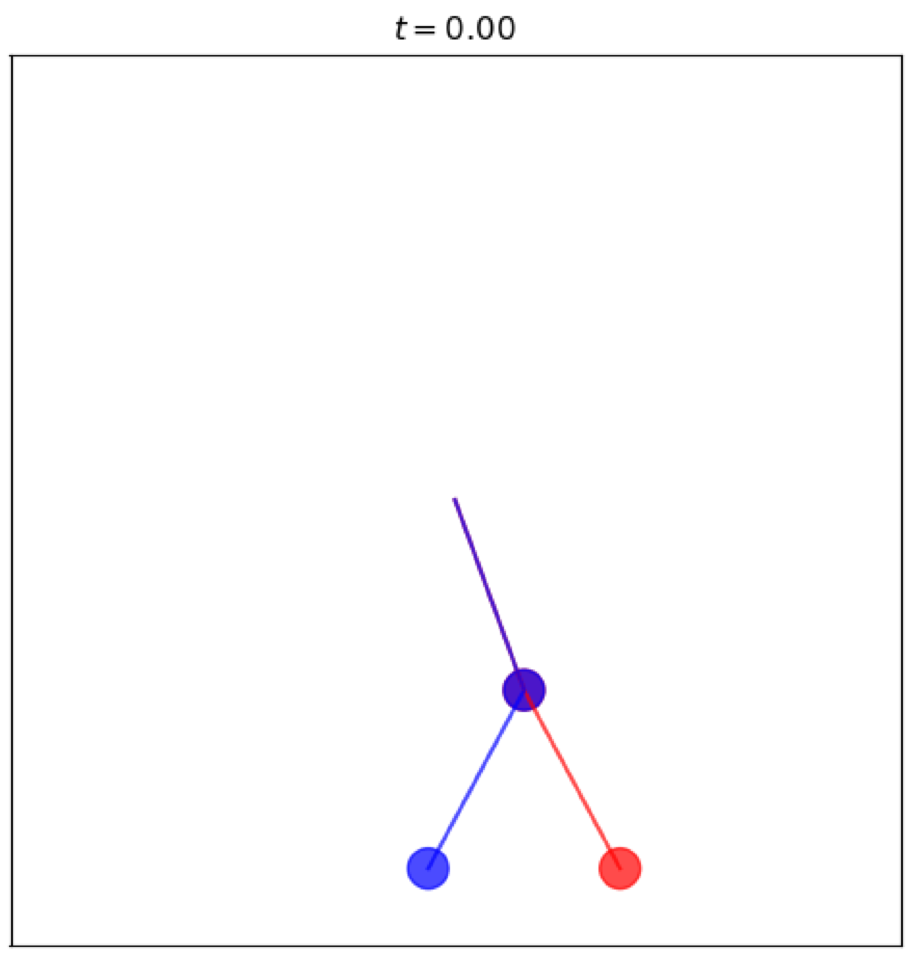
The following cells contains a visualisation function that animates the double pendulum given a list of times and state vectors. You might find it useful.

```
In [2]: %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 [3]:
```
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

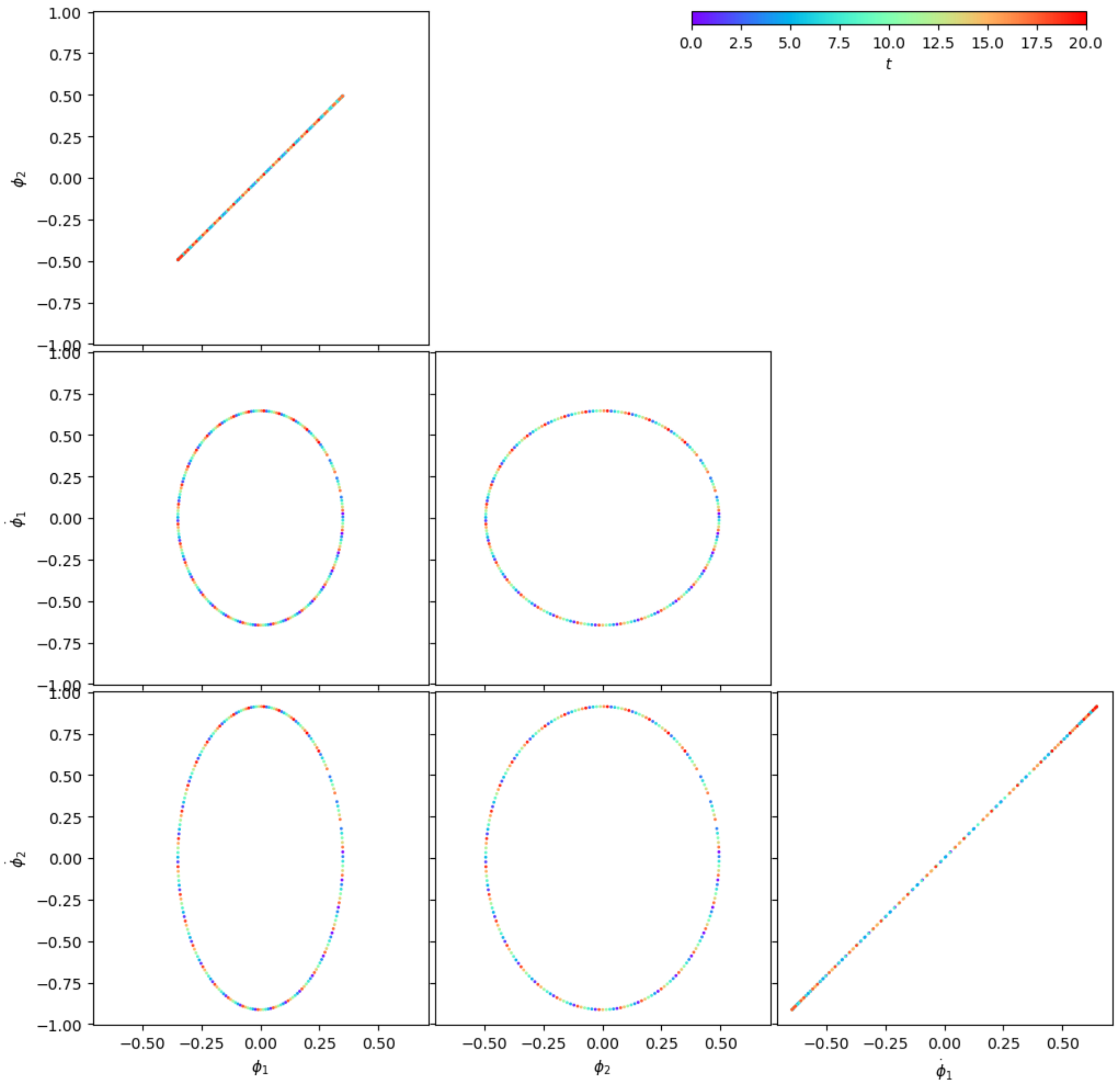
1f.

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

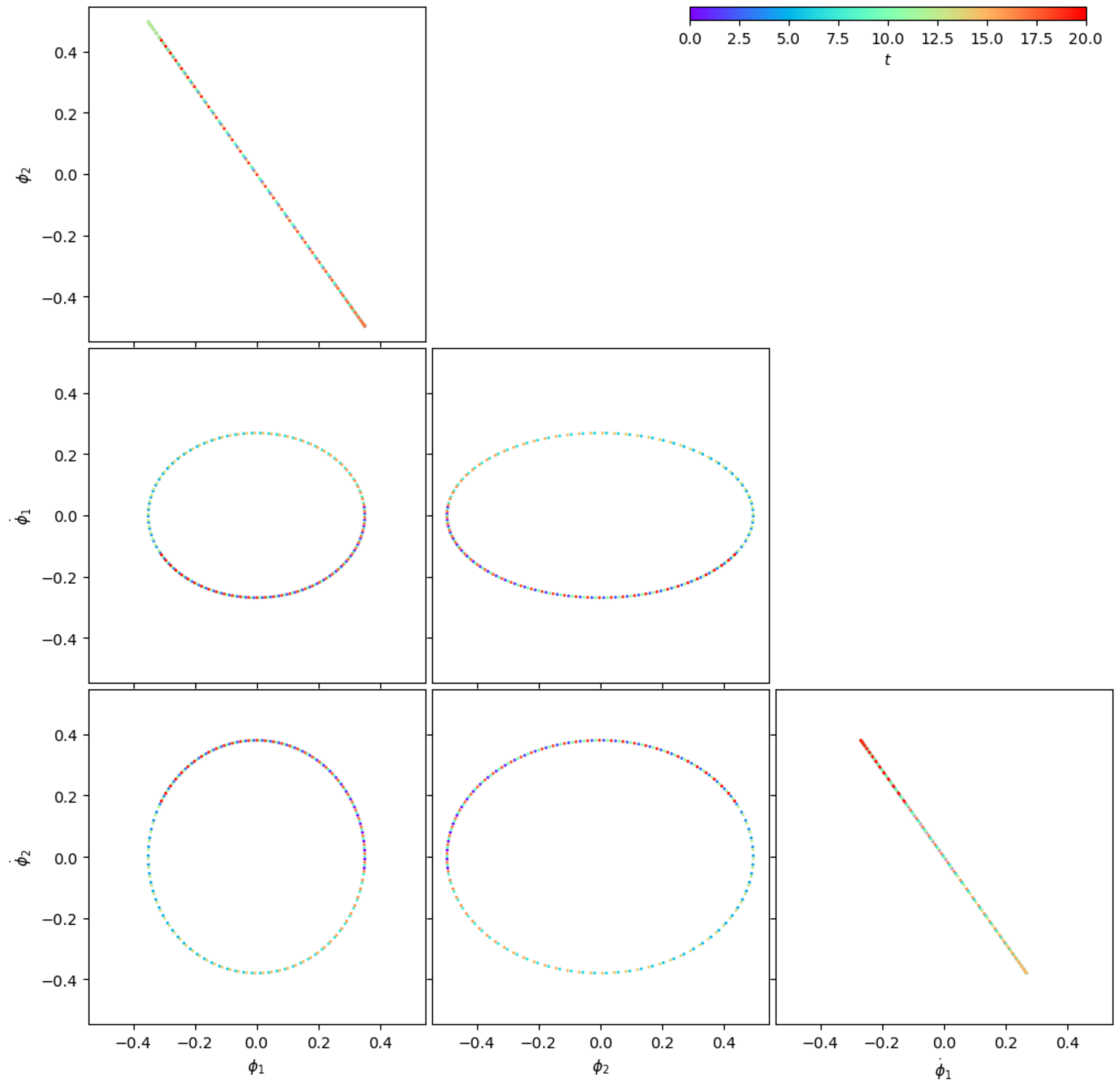
In [4]:
```python
from matplotlib.colors import Normalize
from matplotlib.colorbar import ColorbarBase
from matplotlib import cm as cmaps

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 [5]:
```
plot_phase_space(times, normal_mode_1)
```



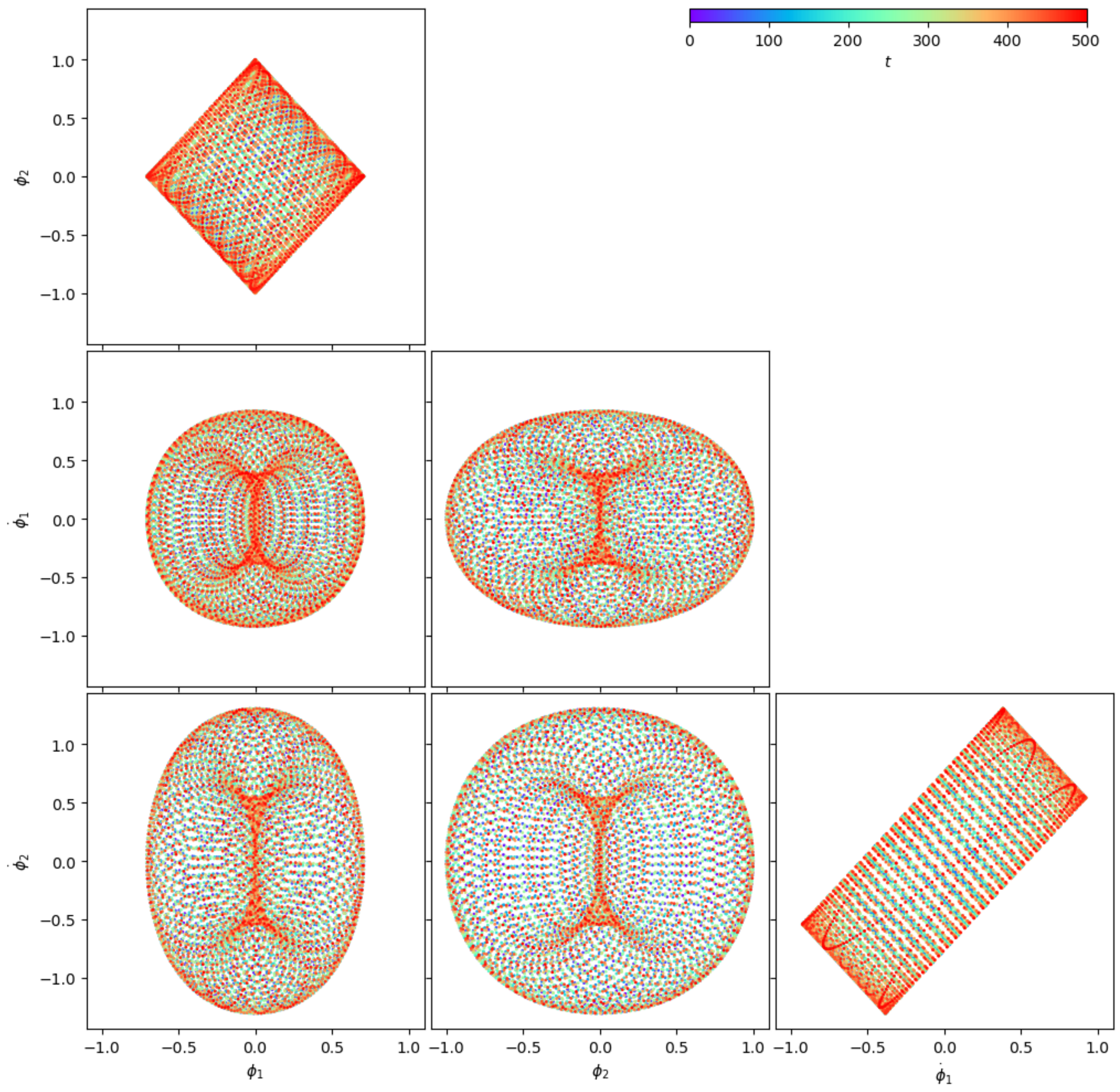
In [6]:
```
plot_phase_space(times, normal_mode_2)
```



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 [7]:
```
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:

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

And similarly,

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

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 [32]: 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
             phi1 = state[0]
             phi2 = state[1]
             phidot1 = state[2]
             phidot2 = state[3]
```

```
    A1 = (m1 + m2)* L1**2
    B1 = m2 * L1 * L2 * np.cos(phi2 - phi1)
    B2 = m2 * L2**2
    C1 = m2 * L1 * L2 * phidot2**2 * np.sin(phi2 - phi1) - (m1 + m2) * g * L
    C2 = -m2 * L1 * L2 * phidot1**2 * np.sin(phi2 - phi1) - m2*g*L2 * np.sir
    der = np.array([phidot1, phidot2, (B2 * C1 - B1 * C2)/(A1*B2 - B1**2), (
    return der
```

1i.

Solve the EoMs analytically.

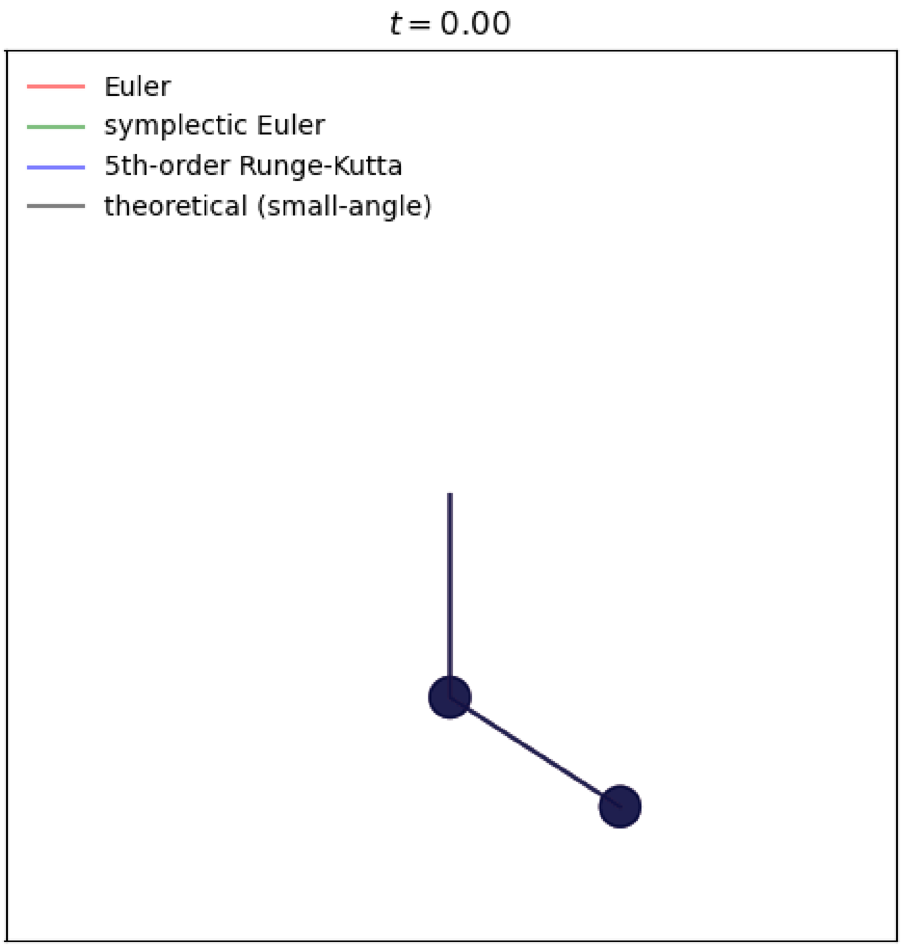
Lol jk, let's use some differential equation solvers to solve the full, non-linear EoMs which we can't solve analytically. The functions given below are implementations of the Euler, Symplectic Euler, and a 5th order Runge-Kutta method (these methods are introduced in the introductory Python notebooks) 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.

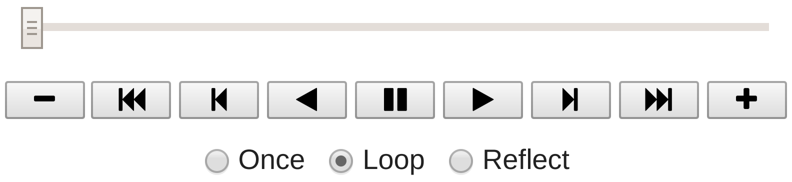
In [9]:
```
from scipy.integrate import solve_ivp

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 [10]:
```
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==True:
        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")
```

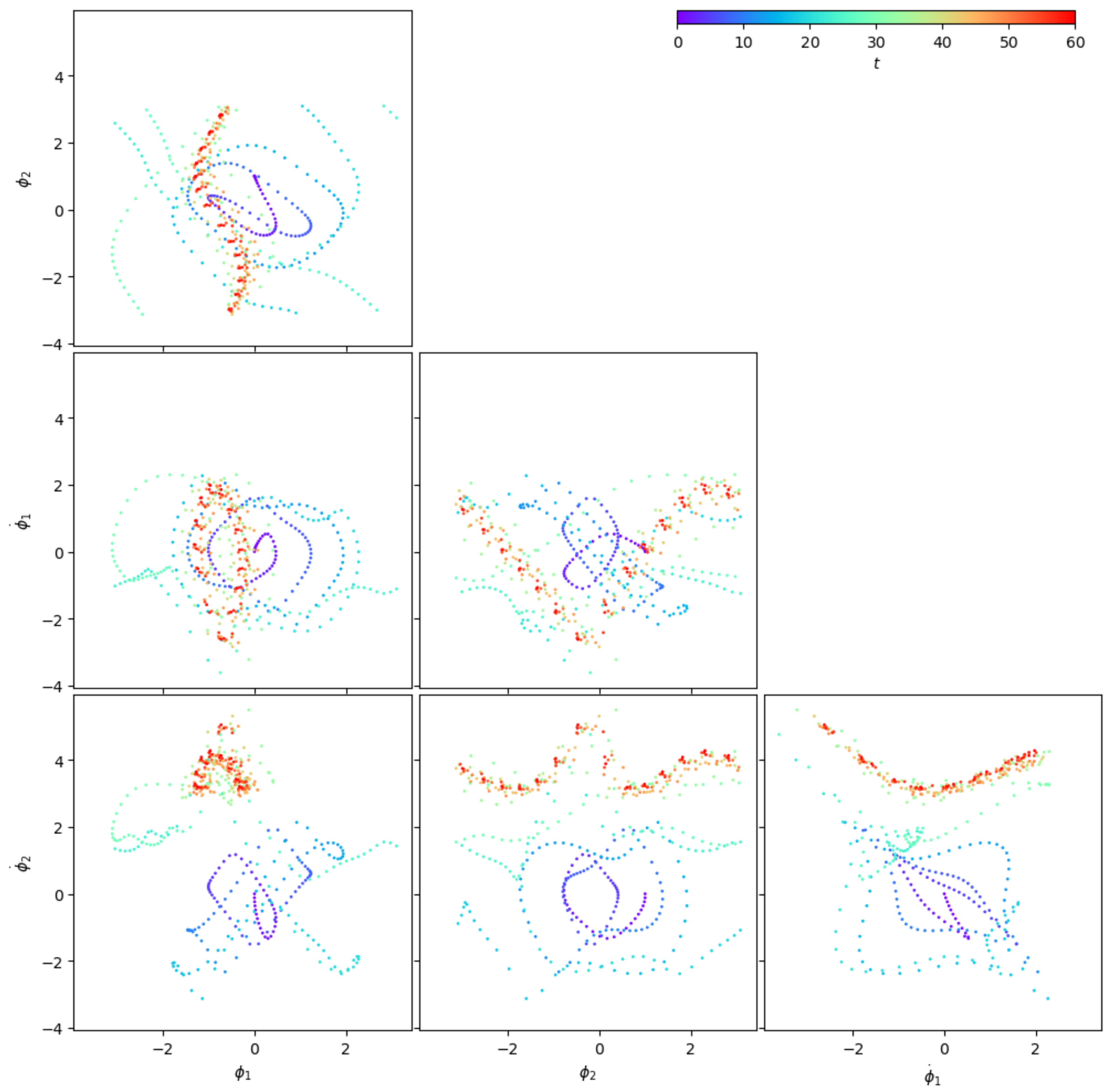
In [11]:
```
%time compare_methods([0,1,0,0], 60, 500, theo=True)
```





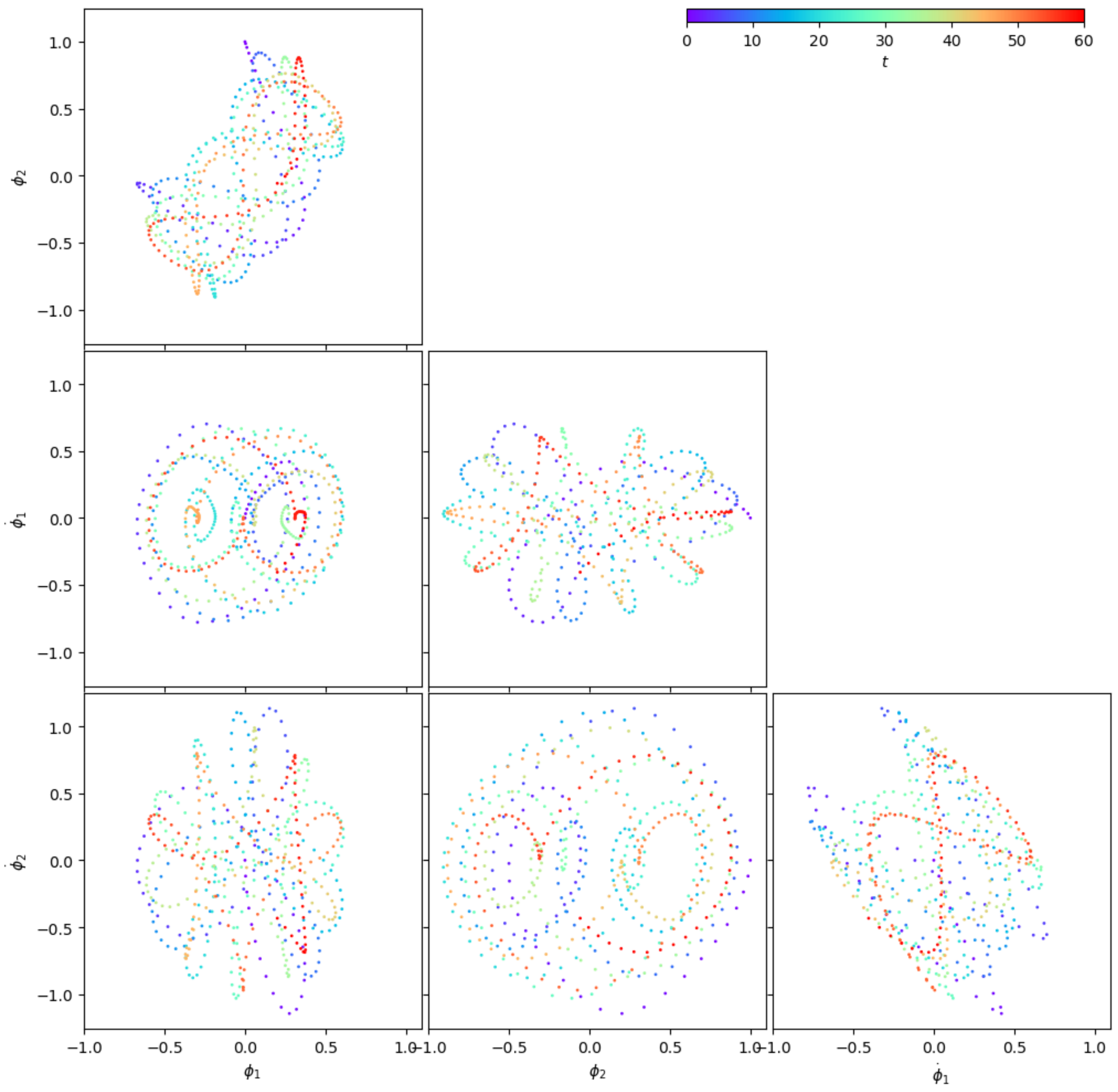
```
CPU times: user 26.6 s, sys: 701 ms, total: 27.3 s
Wall time: 27.3 s
```

Euler

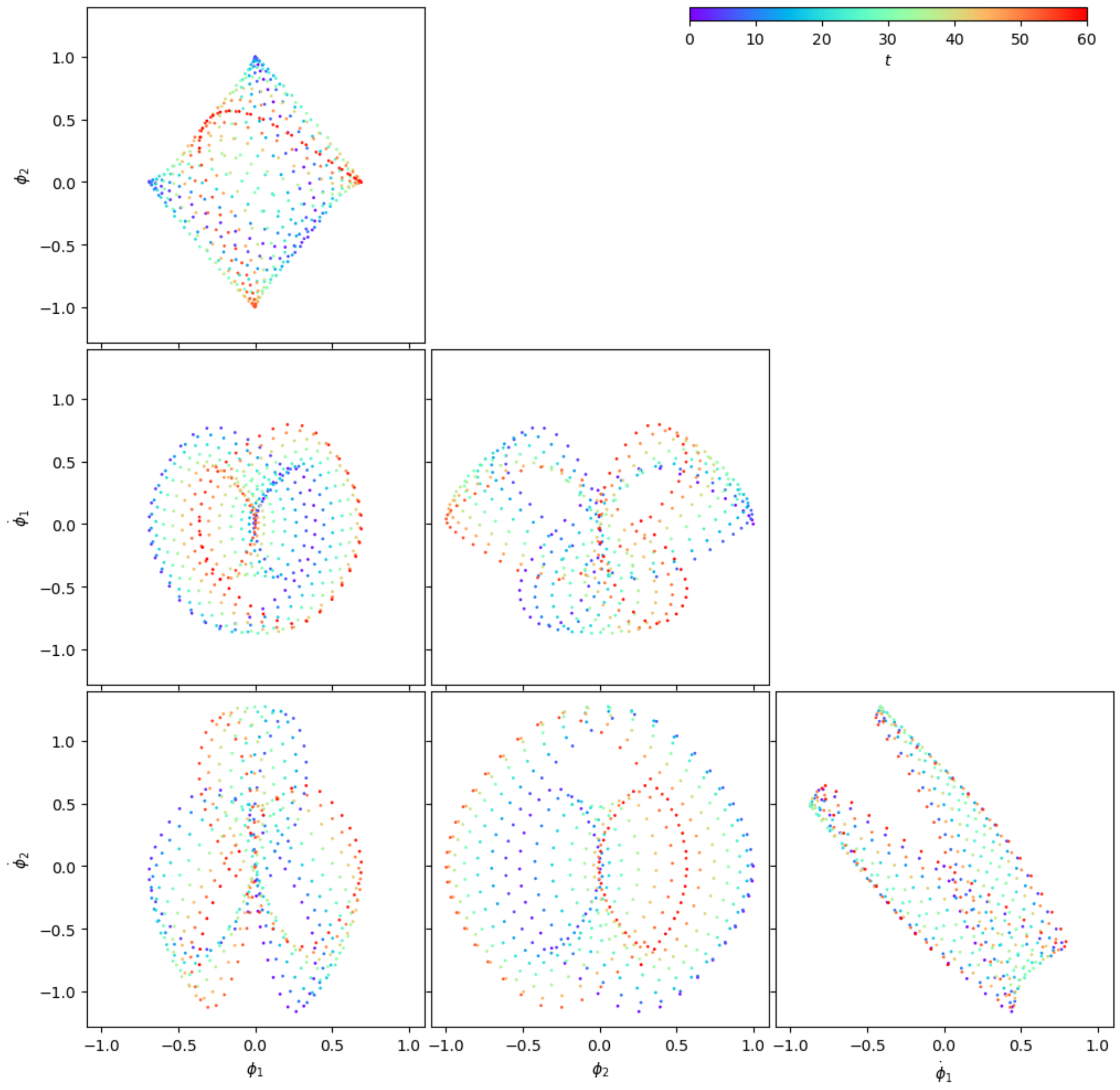


0
10
20
30
40
50
60
t
ϕ_2
$\dot{\phi}_1$
$\dot{\phi}_2$
ϕ_1
ϕ_2
$\dot{\phi}_1$
−4
−2
0
2
4

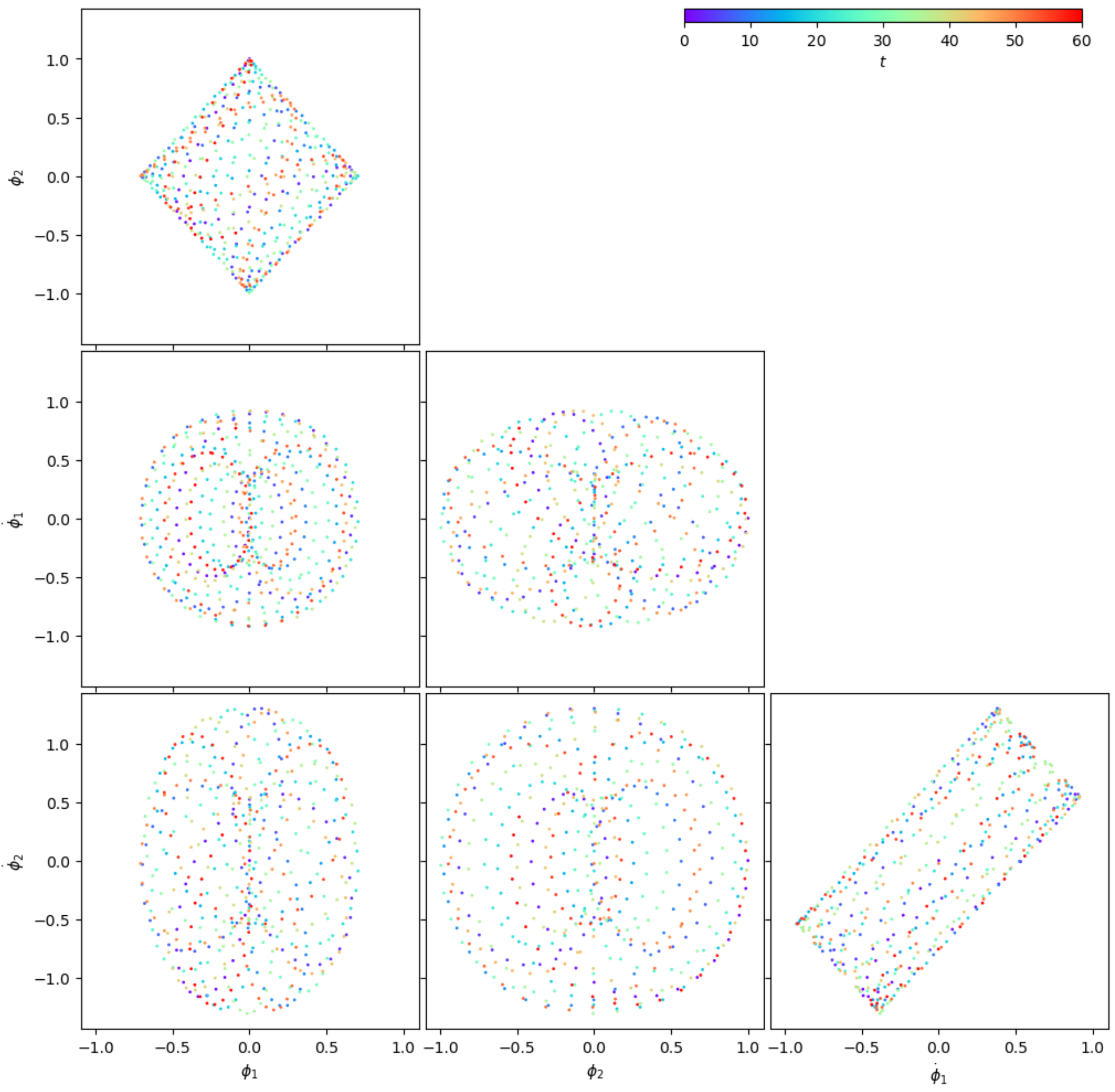
symplectic Euler



5th-order Runge-Kutta



theoretical (small-angle)

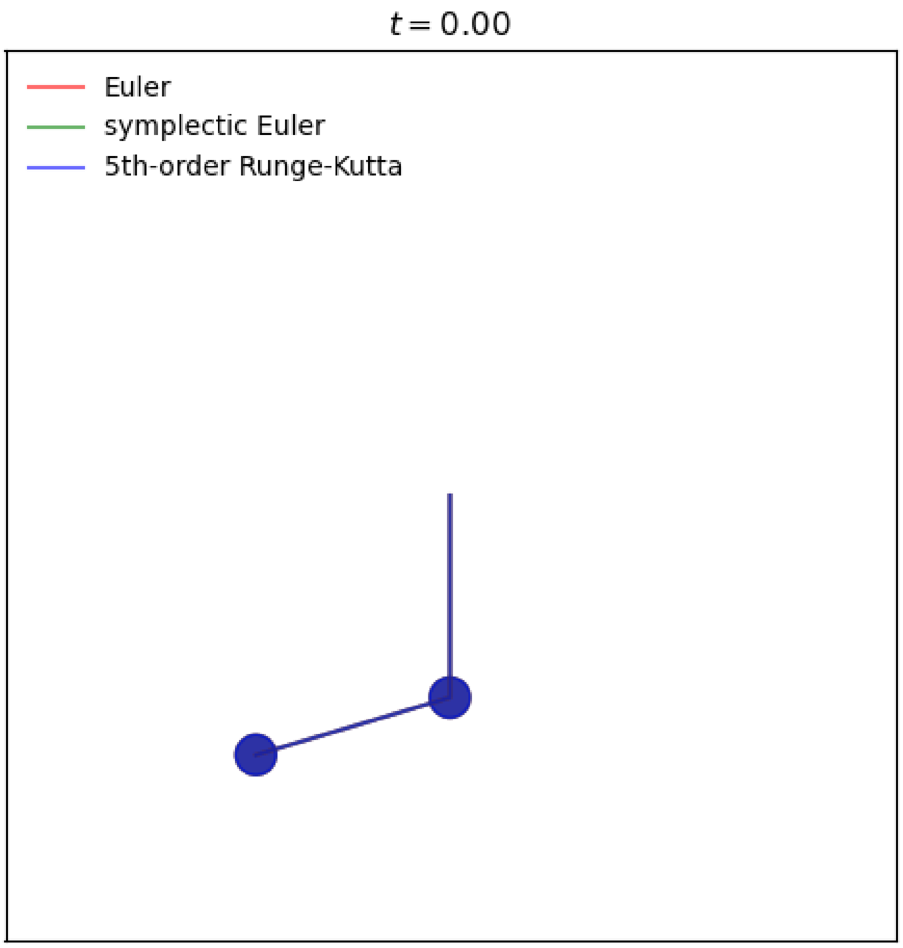


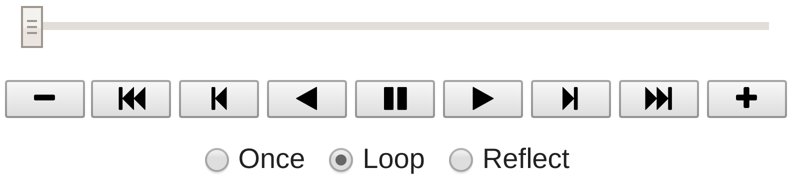
It seems that the Euler method loses track of the system over time, and diverges away from the actual motion the fastest. The symplectic Euler and Runge-Kutta methods both seem to do equally well when compared against one another, but its calculated motion is different than that of the theoretical small 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.

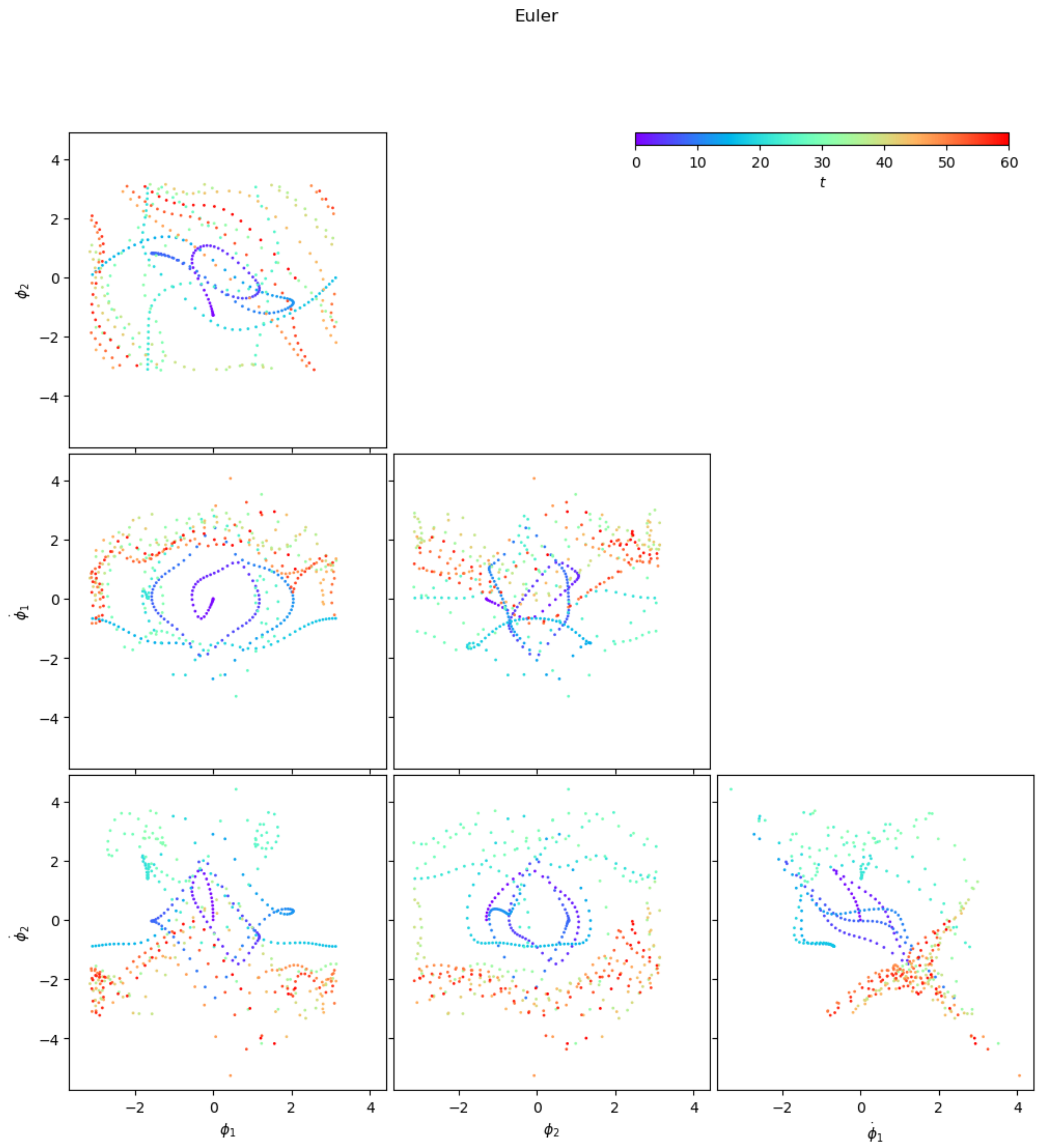
Compare the three methods outside of the small-angle regime. For how long do they display qualitatively similar behaviour? Which method, if any, can we trust?

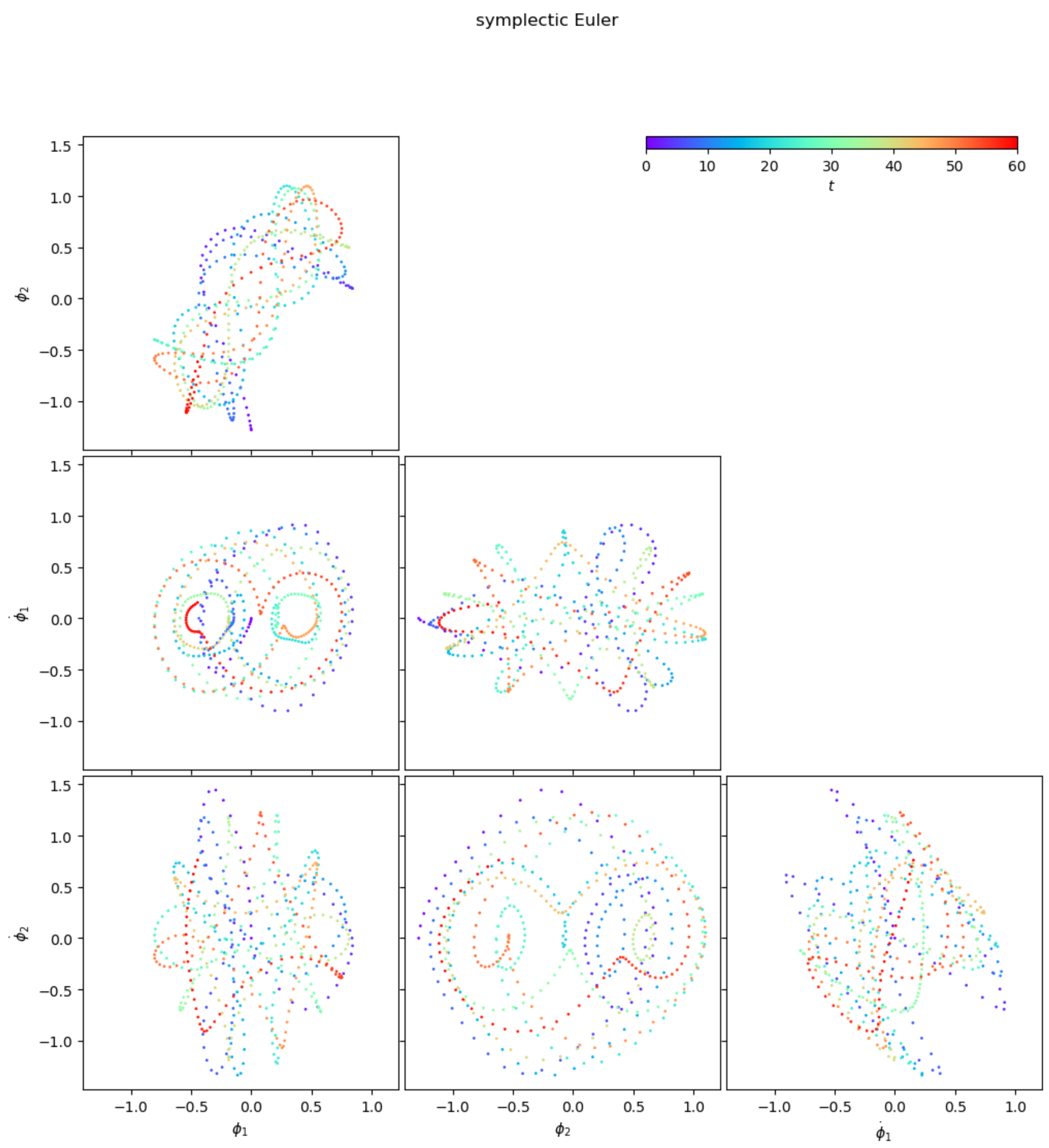
```
In [12]: #Write your answer here
         %time compare_methods([0, 5, 0, 0], 60, 500)
```

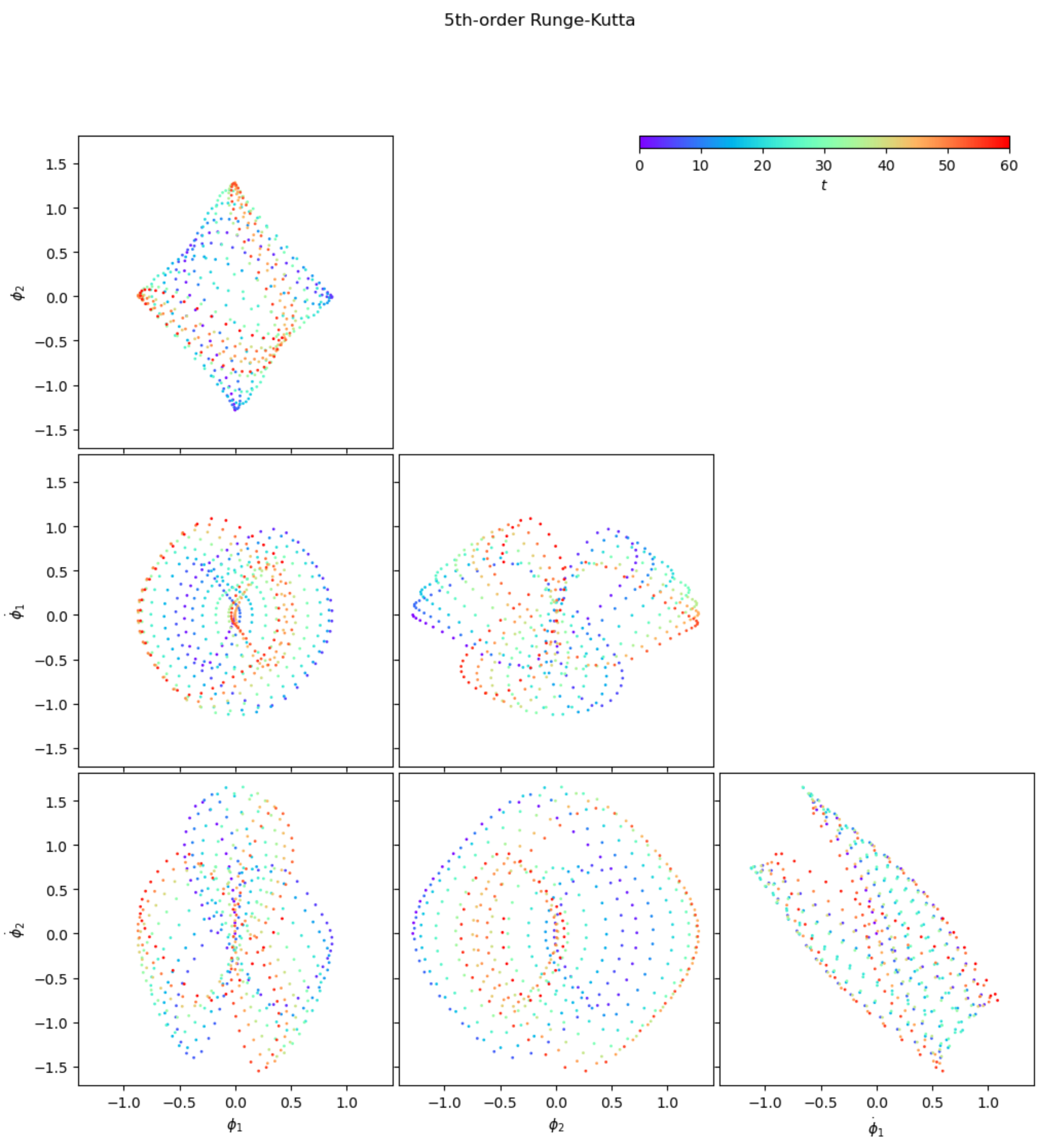




```
CPU times: user 24.1 s, sys: 482 ms, total: 24.6 s
Wall time: 24.6 s
```


Euler
t
0
10
20
30
40
50
60
ϕ_2
$\dot{\phi}_1$
$\dot{\phi}_2$
ϕ_1
ϕ_2
$\dot{\phi}_1$


symplectic Euler
t
0
10
20
30
40
50
60
ϕ_2
$\dot{\phi}_1$
$\dot{\phi}_2$
ϕ_1
ϕ_2
$\dot{\phi}_1$



Increasing the angle, we see that the Euler approximation deviates away from the other two approximations very quickly. On the other hand, it seems that Runge-Kutta and symplectic Euler stay close to one another relatively well - however, it does appear at some large times (approx. $t > 30$), symplectic Euler and Runge-Kutta also start to deviate and oscillate out of phase from one another.

Also, I find that Runge-Kutta seems to give equations of motion that most closely resembles what I think the system should behave.

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?

We aren't considering any external forces to the system, so it makes sense that energy must be conserved. Since gravity, a nonradial force acts on the system, angular momentum cannot be conserved. Another way to see this is the fact that at the maximum height of the system (when the system has only potential energy), its angular velocity is zero, so therefore it has zero angular momentum. On the other hand, we see that when either of the $\phi = 0$, then the angular momentum is large - indicating that L is not conserved.

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 [29]:
```
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
    phi1s = states[0]
    phi2s = states[1]
    phidot1s = states[2]
    phidot2s = states[3]
    result = 1/2 * (m1 + m2) * L1**2 * phidot1s**2 + 1/2 * m2 * L2**2 * phid
    return result
```

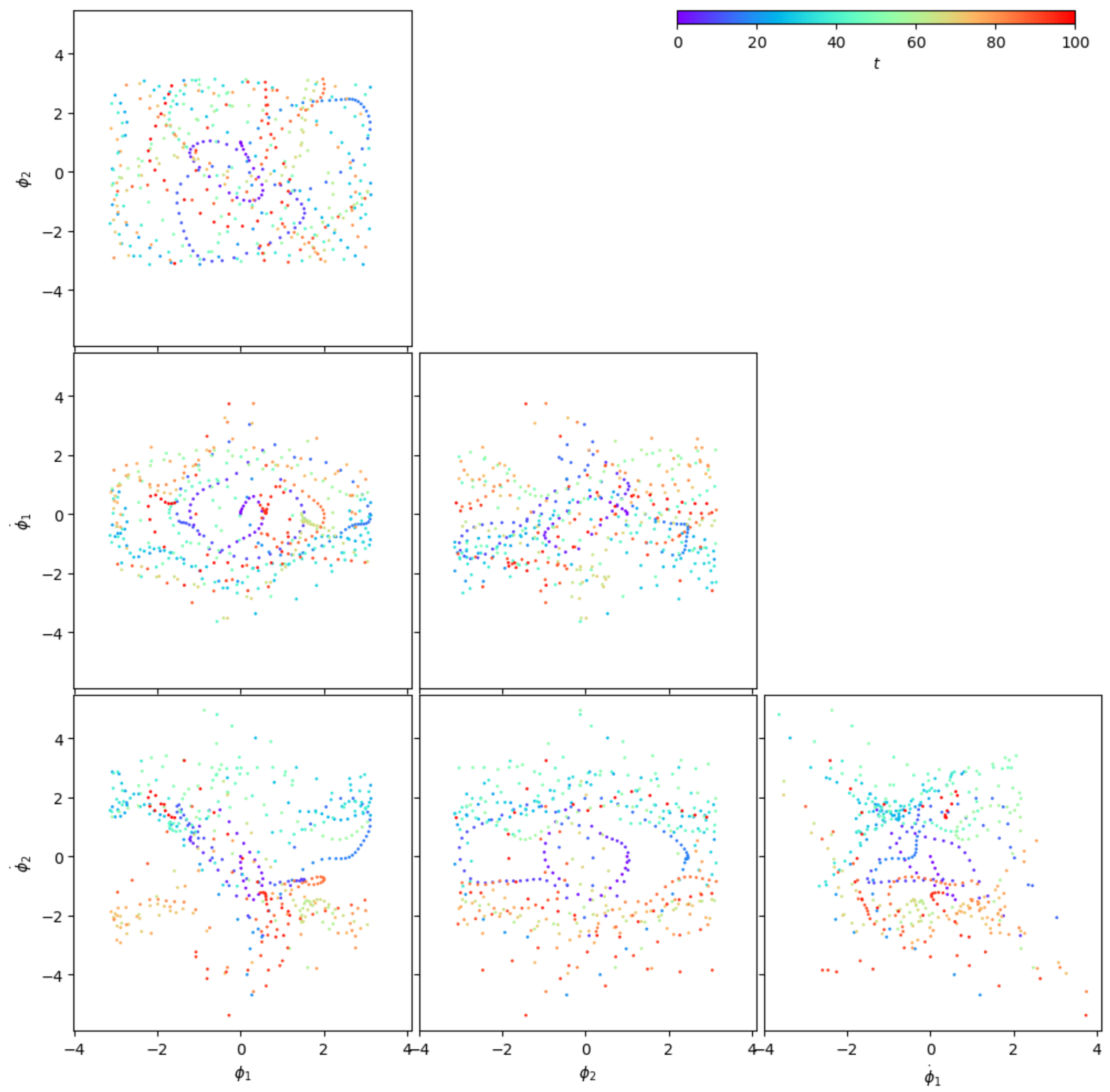
Here, we see that the only method which preserves constant energy is the Runge-Kutta method.

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?

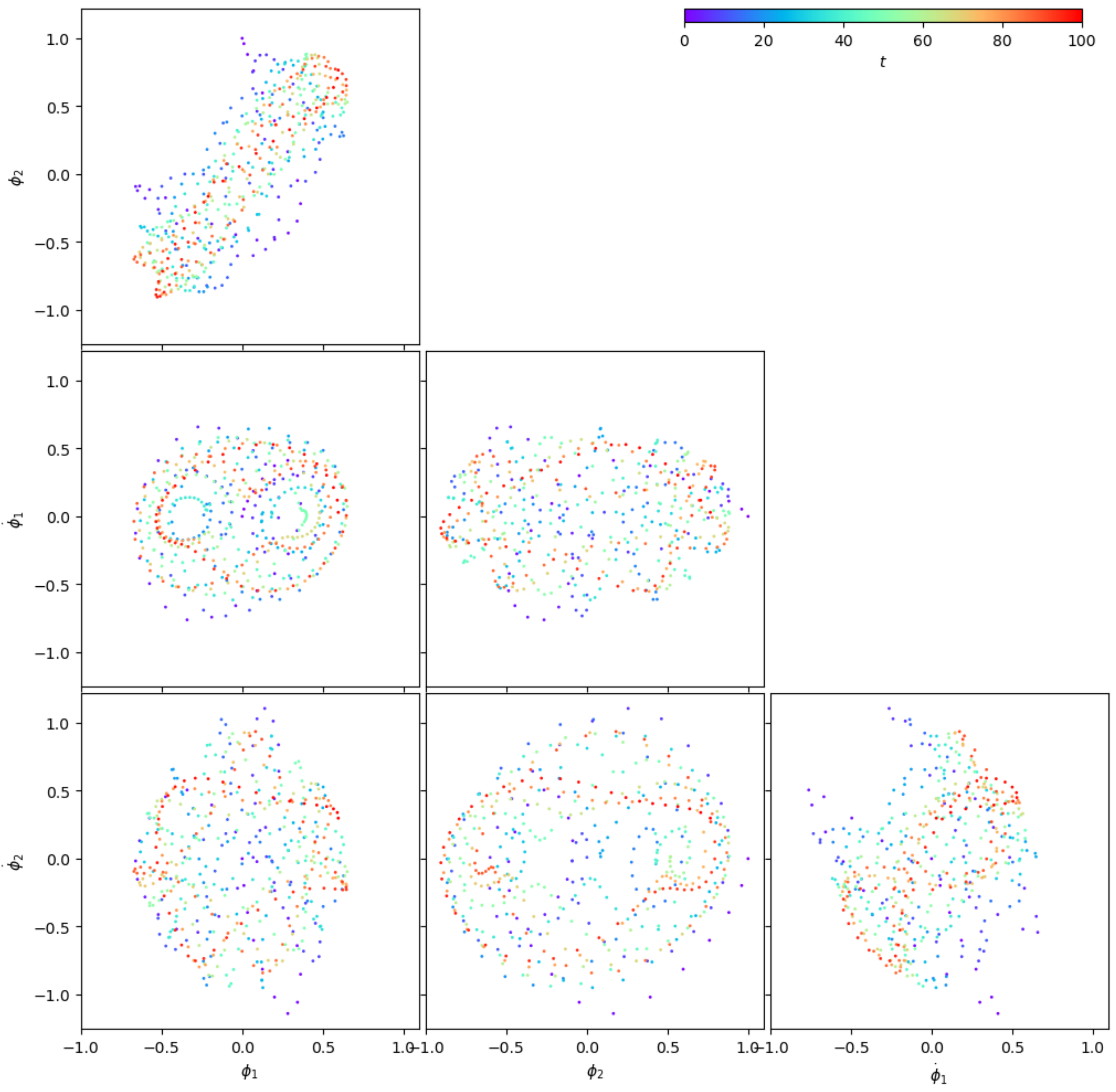
In [33]:
```
#Write your answer here
compare_methods([0,1,0,0], 100, 500, energies=True, anim=False)
```

Euler

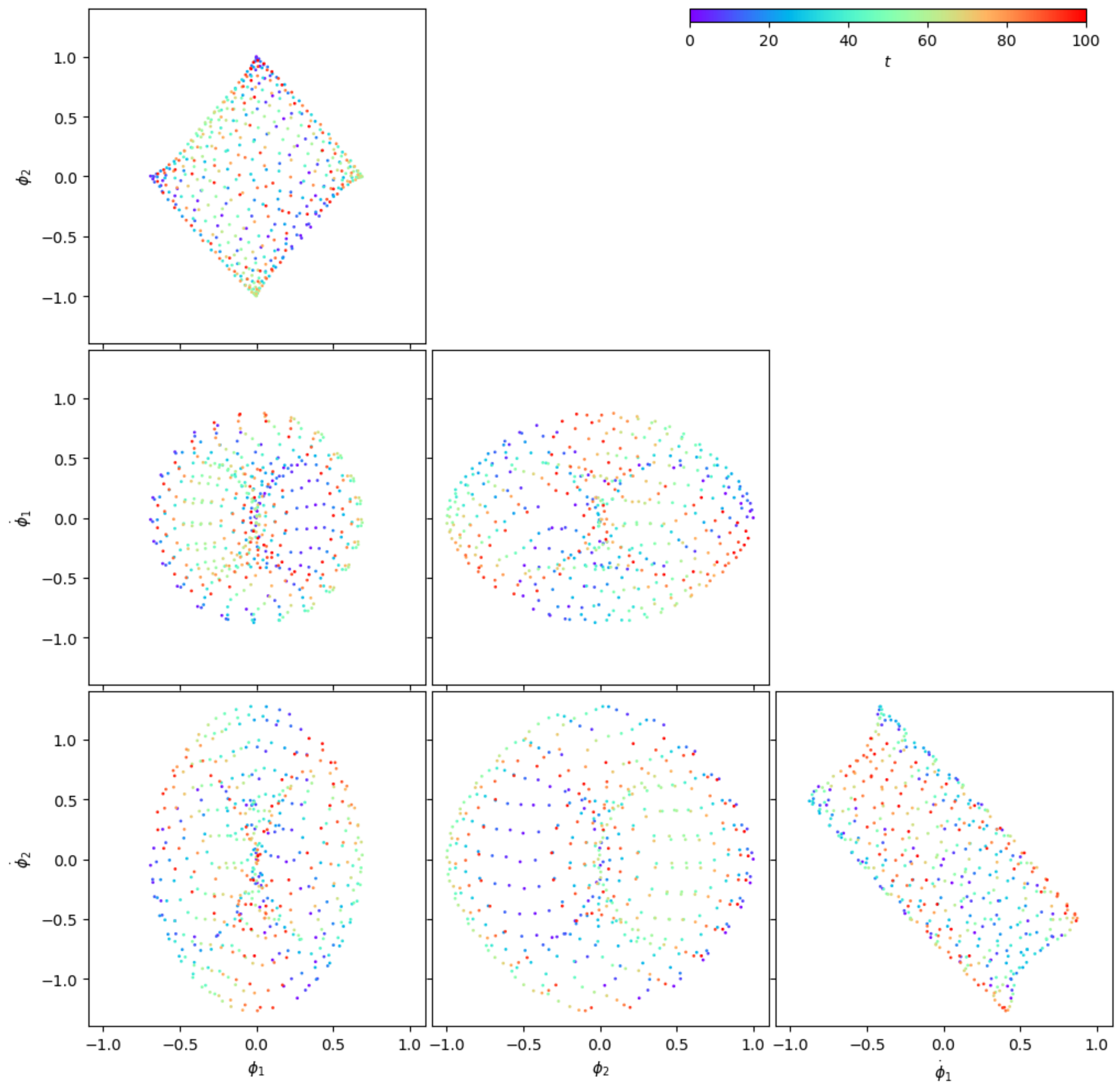


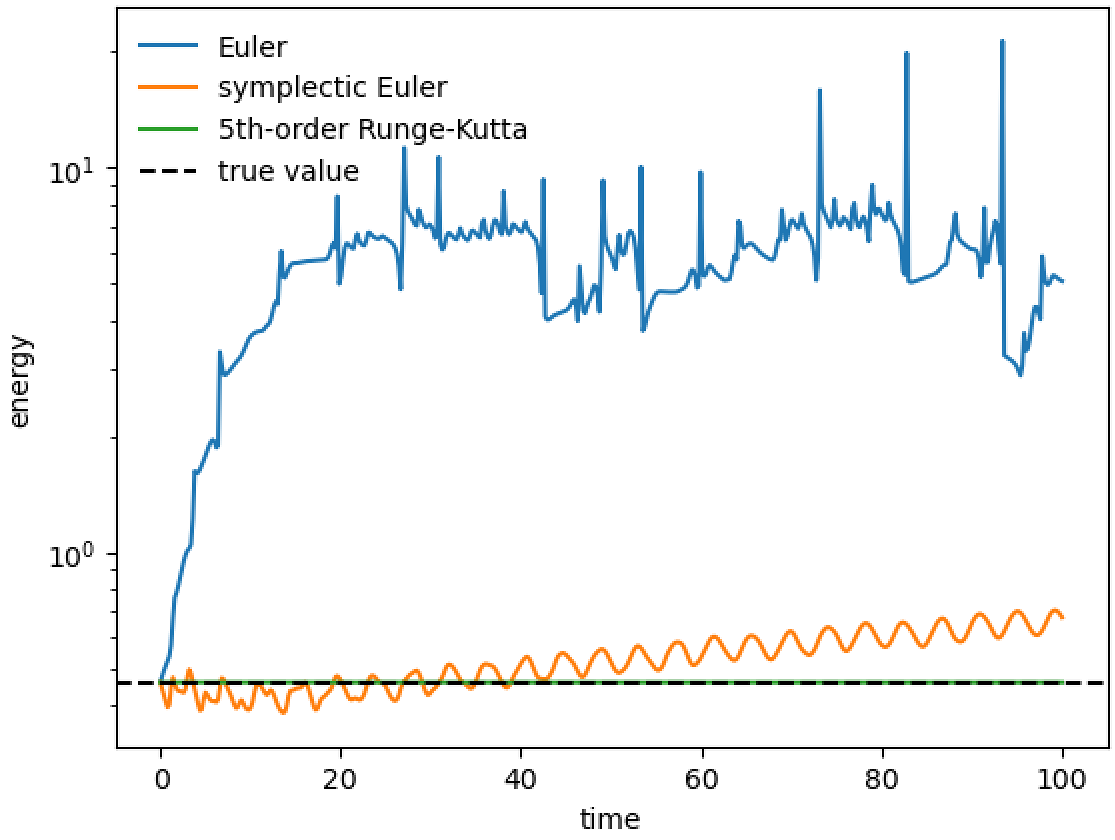
ϕ_2
$\dot{\phi}_1$
$\dot{\phi}_2$
ϕ_1
ϕ_2
$\dot{\phi}_1$
t
0
20
40
60
80
100
4
2
0
−2
−4

symplectic Euler



5th-order Runge-Kutta





We see that the only method which preserves the fact that energy is constant is the Runge-Kutta method, meaning that at large times, the Runge-Kutta method is the only one that we can trust. As far as I could compute, the Runge-Kutta method seemed to consistently return a constant energy, so in principle I think we can trust it for all time.

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 [15]:
```
#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 [16]:
```
#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