Double Pendulum and Multiple Pendulum

CHEN Zehan

June 5, 2021

1 Description

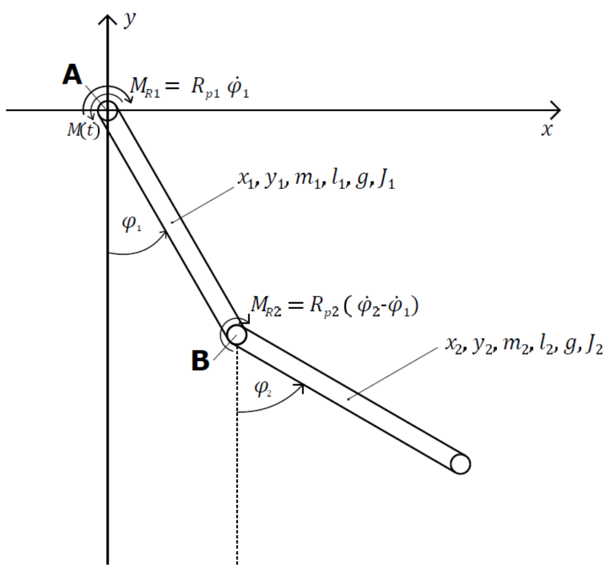

Figure 1: Schematic of a double pendulum

A double pendulum is a combination of two simple pendulums, one attached to the end of another.

We describe the system using R_0, R_1, the lengths of two pendulums, m_0, m_1, the masses of two ends, φ_0, φ_1, the angles of the rods.

We will derive the dynmanical equations and simulate the dynmanics of the system.

2 Dynmanics

We write down the kinetic energy and potential of the system,

$$\begin{aligned}
K &= \frac{1}{2} m_0 \left(R_0 \dot{\varphi}_0\right)^2 + \frac{1}{2} m_1 \left(\left(R_0 \dot{\varphi}_0 \cos\varphi_0 + R_1 \dot{\varphi}_1 \cos\varphi_1\right)^2 + \left(R_0 \dot{\varphi}_0 \sin\varphi_0 + R_1 \dot{\varphi}_1 \sin\varphi_1\right)^2 \right) \\
&= \frac{1}{2} m_0 \left(R_0 \dot{\varphi}_0\right)^2 + \frac{1}{2} m_1 \left(\left(R_0 \dot{\varphi}_0\right)^2 + \left(R_1 \dot{\varphi}_1\right)^2 + 2 R_0 R_1 \dot{\varphi}_0 \dot{\varphi}_1 \cos\left(\varphi_0 - \varphi_1\right) \right) \\
V &= -g m_0 R_0 \cos\varphi_0 - g m_1 R_0 \cos\varphi_0 - g m_1 R_1 \cos\varphi_1 \\
\mathcal{L} &= K - V
\end{aligned}$$

Compute each partial derivatives of Lagrangian,

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \dot{\varphi}_0} &= \frac{\partial K}{\partial \dot{\varphi}_0} = m_0 R_0^2 \dot{\varphi}_0 + m_1 \left(R_0^2 \dot{\varphi}_0 + R_0 R_1 \dot{\varphi}_1 \cos\left(\varphi_0 - \varphi_1\right) \right) \\
\frac{\partial \mathcal{L}}{\partial \dot{\varphi}_1} &= \frac{\partial K}{\partial \dot{\varphi}_1} = m_1 \left(R_1^2 \dot{\varphi}_1 + R_0 R_1 \dot{\varphi}_0 \cos\left(\varphi_0 - \varphi_1\right) \right) \\
\frac{\partial \mathcal{L}}{\partial \varphi_0} &= -m_1 R_0 R_1 \dot{\varphi}_0 \dot{\varphi}_1 \sin\left(\varphi_0 - \varphi_1\right) - g m_0 R_0 \sin\varphi_0 - g m_1 R_0 \sin\varphi_0 \\
\frac{\partial \mathcal{L}}{\partial \varphi_1} &= m_1 R_0 R_1 \dot{\varphi}_0 \dot{\varphi}_1 \sin\left(\varphi_0 - \varphi_1\right) - g m_1 R_1 \sin\varphi_1 \\
\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial \mathcal{L}}{\partial \dot{\varphi}_0} &= m_0 R_0^2 \ddot{\varphi}_0 + m_1 \left(R_0^2 \ddot{\varphi}_0 + R_0 R_1 \ddot{\varphi}_1 \cos\left(\varphi_0 - \varphi_1\right) - R_0 R_1 \dot{\varphi}_1 \left(\dot{\varphi}_0 - \dot{\varphi}_1\right) \sin\left(\varphi_0 - \varphi_1\right) \right) \\
\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial \mathcal{L}}{\partial \dot{\varphi}_1} &= m_1 \left(R_1^2 \ddot{\varphi}_1 + R_0 R_1 \ddot{\varphi}_0 \cos\left(\varphi_0 - \varphi_1\right) - R_0 R_1 \dot{\varphi}_0 \left(\dot{\varphi}_0 - \dot{\varphi}_1\right) \sin\left(\varphi_0 - \varphi_1\right) \right)
\end{aligned}$$

Consider Euler-Lagrange equation,

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial \mathcal{L}}{\partial \dot{\varphi}_k} - \frac{\partial \mathcal{L}}{\partial \varphi_k} = 0$$

$$\begin{gathered}
(m_0 + m_1) R_0 \ddot{\varphi}_0 + m_1 R_1 \cos\left(\varphi_0 - \varphi_1\right) \ddot{\varphi}_1 = -m_1 R_1 \dot{\varphi}_1^2 \sin\left(\varphi_0 - \varphi_1\right) - g(m_0 + m_1) \sin\varphi_0 \\
m_1 R_0 \cos\left(\varphi_0 - \varphi_1\right) \ddot{\varphi}_0 + m_1 R_1 \ddot{\varphi}_1 = m_1 R_0 \dot{\varphi}_0^2 \sin\left(\varphi_0 - \varphi_1\right) - g m_1 \sin\varphi_1
\end{gathered}$$

Since $\varphi_0, \varphi_1, \dot{\varphi}_0, \dot{\varphi}_1$ are known, this is a linear equation to $\ddot{\varphi}_0, \ddot{\varphi}_1$,

$$\begin{aligned}
a \ddot{\varphi}_0 + b \ddot{\varphi}_1 &= e \\
c \ddot{\varphi}_0 + d \ddot{\varphi}_1 &= f
\end{aligned}$$

where

$$\begin{aligned}
a &= (m_0 + m_1) R_0 \\
b &= m_1 R_1 \cos\left(\varphi_0 - \varphi_1\right) \\
c &= m_1 R_0 \cos\left(\varphi_0 - \varphi_1\right) \\
d &= m_1 R_1 \\
e &= -m_1 R_1 \dot{\varphi}_1^2 \sin\left(\varphi_0 - \varphi_1\right) - g(m_0 + m_1) \sin\varphi_0 \\
f &= m_1 R_0 \dot{\varphi}_0^2 \sin\left(\varphi_0 - \varphi_1\right) - g m_1 \sin\varphi_1
\end{aligned}$$

The solution is,

$$\ddot{\varphi}_0 = \frac{de - bf}{ad - bc}$$
$$\ddot{\varphi}_1 = \frac{af - ce}{ad - bc}$$

3 Program

The program is straightforward. We define a class called double pendulum and define functions to compute Lagrangian and solution as shown above.

```python
# xi = [phi_0, phi_1, phi_dot_0, phi_dot_1]
g = 9.8

class double_pendulum:
    def __init__(self, R, M, Phi0, ode = rk4) -> None:
        self.R = R # length
        self.M = M # mass
        self.Phi0 = Phi0 # initial angles
        self.xi0 = np.array([Phi0[0], Phi0[1], 0, 0])
        self.ode = ode # need to appoint a ode solver
        self.step = 0.01 # default step

    def diff_xi(self, xi, t): # compute d(xi)/dt
        diff_xi = np.array([xi[2], xi[3], 0, 0])
        phi = np.array([xi[0], xi[1]])
        phi_dot = np.array([xi[2], xi[3]])
        m = self.M
        R = self.R

        a = (m[0]+m[1])*(R[0]**2)
        b = c = m[1]*R[0]*R[1]*np.cos(phi[0]-phi[1])
        d = m[1]*(R[1]**2)
        e = -m[1]*R[0]*R[1]*(phi_dot[1]**2)*np.sin(phi[0]-phi[1])-g*(m[0]+m[1])*R[0]*np.sin(phi[0])
        f = m[1]*R[0]*R[1]*(phi_dot[0]**2)*np.sin(phi[0]-phi[1])-g*m[1]*R[1]*np.sin(phi[1])

        D = a*d - b*c
        diff_xi[2] = (d*e - b*f) / D
        diff_xi[3] = (a*f - c*e) / D
        return diff_xi

    def Ek(self, xi): # kinetic energy
        phi = np.array([xi[0], xi[1]])
        phi_dot = np.array([xi[2], xi[3]])
        m = self.M
```

```
        R = self.R
        return m[0]/2*(R[0]*phi_dot[0])**2 + m[1]/2*((R[0]*phi_dot[0])**2 + (R[1]*phi_dot[1])**2 \
            + 2*R[0]*R[1]*phi_dot[0]*phi_dot[1]*np.cos(phi[0] - phi[1]))

    def V(self, xi): # potential
        phi = np.array([xi[0], xi[1]])
        m = self.M
        R = self.R
        return -g*(m[0]*R[0]*np.cos(phi[0]) + m[1]*R[0]*np.cos(phi[0]) + m[1]*R[1]*np.cos(phi[1]))

    def Lagrangian(self, xi):
        return self.Ek(xi) - self.V(xi)

    def Hamiltonian(self, xi):
        return self.Ek(xi) + self.V(xi)

    def simulate(self, t0, t1, h = 0.1):
        T, Xi = self.ode(self.diff_xi, t0, self.xi0, t1, h)
        return T, Xi
```

We write two ODE solvers for comparison.

```
# f: return dx/dt, t0: start time, x0: initial condition, ti: end time, h: step
def rk4(f, t0, x0, ti, h = 1): # 4th-order Runge-Kutta
    T = [t0]
    X = [x0]
    while T[-1] < ti:
        x, t = X[-1], T[-1]
        k1 = h * f(x, t)
        k2 = h * f(x + 0.5*k1, t + 0.5*h)
        k3 = h * f(x + 0.5*k2, t + 0.5*h)
        k4 = h * f(x + k3, t + h)
        T.append(t + h)
        X.append(x + (k1 + 2*k2 + 2*k3 + k4)/6)
    return np.array(T), np.array(X)

def leapfrog(f, t0, x0, ti, h = 1):
    T = [t0]
    X = [x0]
    x_half = x0 + 0.5 * h * f(x0, t0)
    while T[-1] < ti:
        x, t = X[-1], T[-1]
        T.append(t + h)
        X.append(x + h * f(x_half, t + 0.5*h))
        x_half += h * f(X[-1], T[-1])
    return np.array(T), np.array(X)
```

As for plot and animation parts, please refer to the source codes.

4 Performance

We plot total energy using rk4 and leapfrog with smaller steps to see whether they converge.

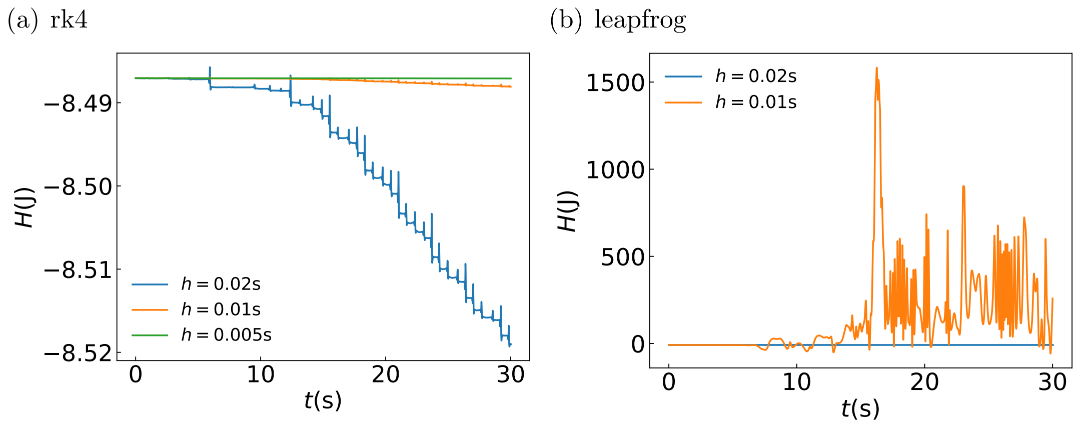

Figure 2: Comparison of convergence between rk4 and leapfrog algorithm. The plot is Hamiltonian versus time. Total energy using rk4 converges but that of leapfrog method does not converge. Initial condition is $R_0 = 2, R_1 = 1, m_0 = 2, m_1 = 1, \varphi_0 = \pi/2, \varphi_1 = \pi/6$, unit m,kg,rad.

We also compare the accuracy using different methods.

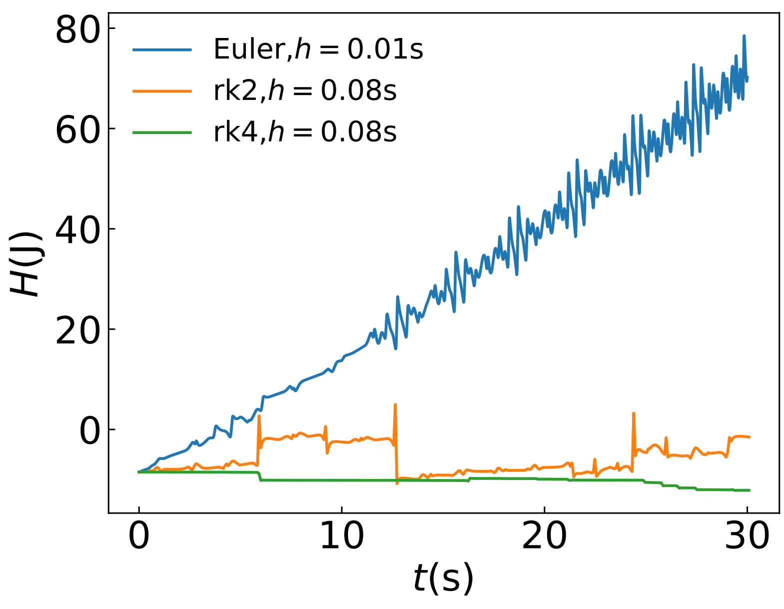

Figure 3: Comparison of accuracy among different methods. Higher order methods have better accuracy even with larger steps. Initial condition is $R_0 = 2, R_1 = 1, m_0 = 2, m_1 = 1, \varphi_0 = \pi/2, \varphi_1 = \pi/6$, unit m,kg,rad.

We plot animations for references. They are attached to the submission.

5 Multiple Pendulum

We write down the Lagrangian and Euler-Lagrange formula for multiple pendulum with n points.

$$
\begin{aligned}
K &= \sum_{i=1}^{n}\left(\frac{1}{2}m_i\left(\left(\sum_{j=1}^{i}R_j\dot{\varphi}_j\cos\varphi_j\right)^2+\left(\sum_{j=1}^{i}R_j\dot{\varphi}_j\sin\varphi_j\right)^2\right)\right)\\
V &= \sum_{i=1}^{n}\sum_{j=1}^{i}-gm_iR_j\cos\varphi_j\\
\mathcal{L} &= K-V\\
\frac{\partial\mathcal{L}}{\partial\dot{\varphi}_k} &= \frac{\partial K}{\partial\dot{\varphi}_k}=\sum_{i=k}^{n}m_i\left(R_k\cos\varphi_k\sum_{j=1}^{i}R_j\dot{\varphi}_j\cos\varphi_j+R_k\sin\varphi_k\sum_{j=1}^{i}R_j\dot{\varphi}_j\sin\varphi_j\right)\\
&= \sum_{i=k}^{n}m_i\sum_{j=1}^{i}R_jR_k\dot{\varphi}_j\cos\left(\varphi_j-\varphi_k\right)\\
\frac{\partial\mathcal{L}}{\partial\varphi_k} &= \sum_{i=l}^{n}m_i\left(-R_k\dot{\varphi}_k\sin\varphi_k\sum_{j=1}^{i}R_j\dot{\varphi}_j\cos\varphi_j+R_k\dot{\varphi}_k\cos\varphi_k\sum_{j=1}^{i}R_j\dot{\varphi}_j\sin\varphi_j\right)\\
&\quad -\sum_{i=k}^{n}gm_iR_k\sin\varphi_k\\
&= \sum_{i=k}^{n}m_i\sum_{j=1}^{i}R_jR_k\dot{\varphi}_j\dot{\varphi}_k\sin\left(\varphi_j-\varphi_k\right)-\sum_{i=k}^{n}gm_iR_k\sin\varphi_k\\
\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{\varphi}_k} &= \sum_{i=k}^{n}m_i\left(\sum_{j=1}^{i}R_jR_k\ddot{\varphi}_j\cos\left(\varphi_j-\varphi_k\right)-\sum_{j=1}^{i}R_jR_k\dot{\varphi}_j\left(\dot{\varphi}_j-\dot{\varphi}_k\right)\sin\left(\varphi_j-\varphi_k\right)\right)
\end{aligned}
$$

$$
\begin{gathered}
\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{\varphi}_k}-\frac{\partial\mathcal{L}}{\partial\varphi_k}=0\\
\sum_{i=k}^{n}m_i\left(\sum_{j=1}^{i}R_jR_k\ddot{\varphi}_j\cos\left(\varphi_j-\varphi_k\right)-\sum_{j=1}^{i}R_jR_k\dot{\varphi}_j^2\sin\left(\varphi_j-\varphi_k\right)+gR_k\sin\varphi_k\right)=0\\
\sum_{i=k}^{n}m_i\left(\sum_{j=1}^{i}R_j\ddot{\varphi}_j\cos\left(\varphi_j-\varphi_k\right)-\sum_{j=1}^{i}R_j\dot{\varphi}_j^2\sin\left(\varphi_j-\varphi_k\right)+g\sin\varphi_k\right)=0
\end{gathered}
$$

Since φ_i and $\dot{\varphi}_i$ are known, it is a linear equation to $\ddot{\varphi}_i$, i.e. $a_k^j\ddot{\varphi}_j=b_k$.

We modify our codes as follows,

```
# xi = [phi_0, phi_1, ..., phi_n; phi_dot_0, phi_dot_1, ..., phi_dot_n]
g = 9.8

class multi_pendulum:
```

```python
    def __init__(self, N, R, M, Phi0, ode = leapfrog, step = 0.01) -> None:
        self.N = N
        self.R = R
        self.M = M
        self.Phi0 = Phi0
        self.xi0 = np.zeros(2*N)
        self.xi0[:N] = np.array(Phi0)
        self.ode = ode
        self.step = step

    def diff_xi(self, xi, t):
        N = self.N
        # xi = [phi0, ..., phi_n; phi_dot_0, ..., phi_dot_n]
        # diff_xi = [phi_dot_0, ..., phi_dot_n; phi_ddot_0, ..., phi_ddot_n]
        diff_xi = np.zeros(2*N)
        diff_xi[:N] = xi[N:]
        phi = xi[:N]
        phi_dot = xi[N:]
        m = self.M
        R = self.R

        a = np.zeros([N, N])
        b = np.zeros(N)
        # summation here follows the equation we already derived
        # the subscripts we use are the same as those in the derivation
        for k in  range(N):
            for i in  range(k, N):
                b[k] += -g * m[i] * np.sin(phi[k])
                for j in  range(i + 1):
                    a[k, j] += m[i] * R[j] * np.cos(phi[j] - phi[k])
                    b[k] += m[i] * R[j] * (phi_dot[j]**2) * np.sin(phi[j] - phi[k])
        # solve linear equation for Ax = b, A = a[k, j], b = b[k]
        diff_xi[N:] = np.linalg.solve(a, b)

        return diff_xi

    def Ek(self, xi):
        N = self.N
        phi = xi[:N]
        phi_dot = xi[N:]
        m = self.M
        R = self.R

        K = 0
        vx = vy = 0
        for i in  range(N):
            vx += R[i] * phi_dot[i] * np.cos(phi[i])
            vy += R[i] * phi_dot[i] * np.sin(phi[i])
```

```
            K += 0.5 * m[i] * (vx**2 + vy**2)

        return K

    def V(self, xi):
        N = self.N
        phi = xi[:N]
        m = self.M
        R = self.R

        V = 0
        m_acc = np. sum(m)
        for i in  range(N):
            V += -g * m_acc * R[i] * np.cos(phi[i])
            m_acc -= m[i]

        return V

    def Lagrangian(self, xi):
        return self.Ek(xi) - self.V(xi)

    def Hamiltonian(self, xi):
        return self.Ek(xi) + self.V(xi)

    def simulate(self, t0, t1, h = 0.1):
        T, Xi = self.ode(self.diff_xi, t0, self.xi0, t1, h)
        return T, Xi
```

Again, we verify the convergence of total energy.

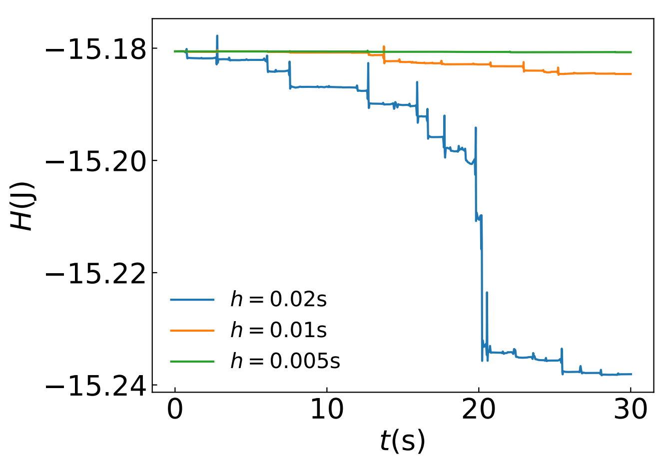

Figure 4: Hamiltonian versus time using rk4 with decreasing steps. With smaller step, the total energy converges. Initial condition is $N = 3, R_0 = 2, R_1 = 1, R_2 = 1, m_0 = 2, m_1 = 1, m_2 = 0.5, \varphi_0 = \pi/2, \varphi_1 = \pi/6, \varphi_2 = -\pi/3$, unit m,kg,rad.

Here we plot some trajectories of the pendulum.

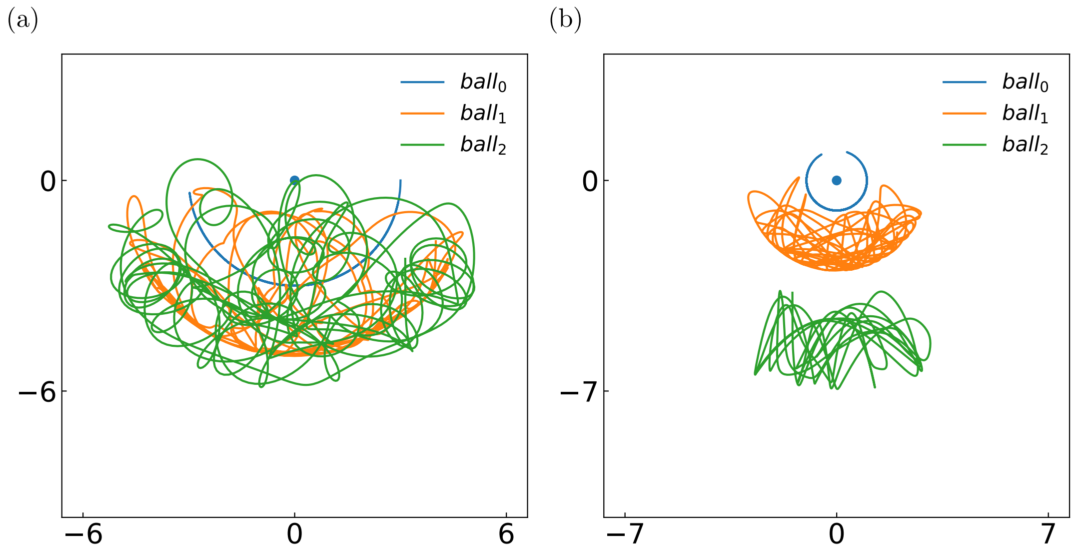

Figure 5: $N = 3$. (a) Initial condition $R_0 = 3, R_1 = 2, R_2 = 1, m_0 = 2, m_1 = 1, m_2 = 0.5, \varphi_0 = \pi/2, \varphi_1 = \pi/6, \varphi_2 = -\pi/3$, unit m,kg,rad. (b) Initial condition $R_0 = 1, R_1 = 2, R_2 = 4, m_0 = 2, m_1 = 1, m_2 = 3, \varphi_0 = \pi/2, \varphi_1 = \pi/6, \varphi_2 = -\pi/3$, unit m,kg,rad.

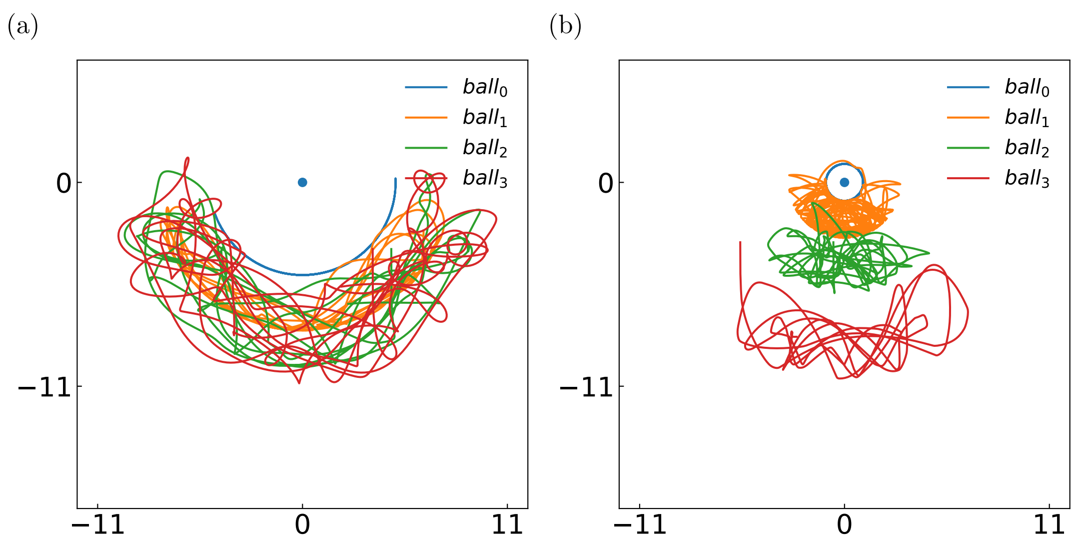

Figure 6: $N = 4$. (a) Initial condition $R_0 = 5, R_1 = 3, R_2 = 2, R_3 = 1, m_0 = 2, m_1 = 1, m_2 = 0.5, m_3 = 0.25, \varphi_0 = \pi/2, \varphi_1 = \pi/6, \varphi_2 = -\pi/3, \varphi_2 = -\pi/2$, unit m,kg,rad. (b) Initial condition $R_0 = 1, R_1 = 2, R_2 = 3, R_3 = 5, m_0 = 1, m_1 = 2, m_2 = 4, m_3 = 8, \varphi_0 = \pi/2, \varphi_1 = \pi/6, \varphi_2 = -\pi/3, \varphi_2 = -\pi/2$, unit m,kg,rad.