循环摆的 Lagrangian 方法

2019 年 5 月 28 日

1 题目回顾

Problem No.14 "Looping pendulum"

Connect two loads, one heavy and one light, with a string over a horizontal rod and lift up the heavy load by pulling down the light one. Release the light load and it will sweep around the rod, keeping the heavy load from falling to the ground. Investigate this phenomenon.

2 求解方法

我们采取 Lagrangian 方法列出方程。根据力学分析，循环摆的 **Lagrangian 函数**可以表示为

$$L = T - V,$$

其中

$$T = \frac{1}{2}m\left[(\dot{l}+R\dot{\varphi})^2 + l^2\dot{\varphi}^2\right] + \frac{1}{2}M(\dot{l}+R\dot{\varphi})^2,$$
$$V = -MgH + mg(R\sin\varphi + l\cos\varphi),$$

比较难以处理的是摩擦力，应该由**绞盘方程**计算得到，这里直接给出

$$f = M(g + R\ddot{\varphi} + \ddot{l})(1-\mathrm{e}^{-\mu\varphi}).$$

列出 Lagrangian 方程

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{l}}\right) - \frac{\partial L}{\partial l} = f,$$
$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{\varphi}}\right) - \frac{\partial L}{\partial \varphi} = 0,$$

得到

$$(m+M\mathrm{e}^{-\mu\varphi})\ddot{l} + (m+M\mathrm{e}^{-\mu\varphi})R\ddot{\varphi} = ml^2\ddot{\varphi} - Mg\mathrm{e}^{-\mu\varphi} - mg\cos\varphi,$$

$$(m+M)R\ddot{l} + [ml^2 + (m+M)R^2]\ddot{\varphi} = -2ml\dot{l}\dot{\varphi} - mg(R\cos\varphi - l\sin\varphi) - MgR.$$

求解这一微分方程组，以得到第一阶段运动。

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import ode
from scipy.linalg import solve
from numpy import pi
```

[2]:
```
#
g = 980.                #
m = 50.                 #
M = 200.                #
u = 0.1                 #
R = 1                   #
```

[3]:
```
#
#
class Looping_Pendulum(object) :

    def __init__(self, m, M, u, R) :
        self.m, self.M, self.u, self.R = m, M, u, R

    def f(self, t, x) :
        l, phi, vl, vphi = x.tolist()

        Dl = vl
        Dphi = vphi

        A = np.mat([[self.m + self.M*np.exp(-self.u*phi)   ,   (self.m + self.
 ↪M*np.exp(-self.u*phi))*self.R    ],
                    [(self.m + self.M)*self.R              ,   self.
 ↪m*l**2+(self.m + self.M)*self.R**2         ]])
        b = np.mat([[self.m*l*vphi**2 - self.M*g*np.exp(-self.u*phi) - self.
 ↪m*g*np.cos(phi)],
                    [-2*self.m*l*vl*vphi - self.m*g*(self.R*np.cos(phi)-l*np.
 ↪sin(phi)) - self.M*g*self.R ]])
        x = solve(A,b)
        Dvl, Dvphi = x[0], x[1]
        return [Dl, Dphi, Dvl, Dvphi]
```

[4]:
```
#
l_0 = 90.
phi_0 = pi/2
vl_0 = 0.
vphi_0 = 0.

system = Looping_Pendulum(m=m, M=M, u=u, R=R)
r = ode(system.f)

initial = [l_0, phi_0, vl_0, vphi_0]
r.set_initial_value(initial, 0)

r.set_integrator("vode", method="bdf")
dt = 1.0e-3 * R
T = 2
```

3 求解方程

我们考虑一阶段运动的截止条件：1. $\dot{l} + R\dot{\varphi} \leq 0$ 2. $l \geq 0$ 3. $t < T$ （运动截止时间）随后物体进入第二阶段运动。

[5]:
```
t = [0,]
U = [initial,]
while r.successful() and r.t+dt < T and r.y[0] >= 0 and (r.y[2] + R*r.y[3]) <=␣
 ↪0:
    r.integrate(r.t+dt)
    t.append(r.t)
    U.append(r.y)
t = np.array(t)
U = np.array(U)
np.shape(U)
```

[5]:
```
(464, 4)
```

[6]:
```
#
l = U[:,0]
```

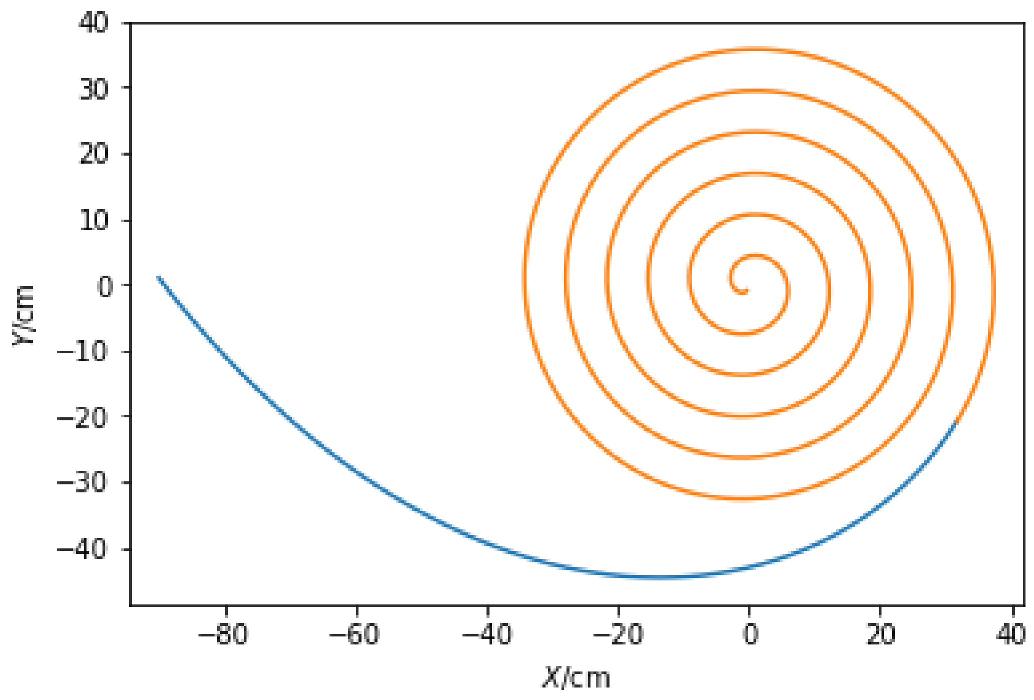
```
phi = U[:,1]
X1 = R*np.cos(phi) - l*np.sin(phi)
Y1 = R*np.sin(phi) + l*np.cos(phi)
```

[7]:
```
#
phi_1 = U[-1,1]
l_1 = U[-1,0]
phi_2 = phi_1 + l_1 / R
phi_p = np.linspace(phi_1, phi_2, 1000)
l_p = (phi_2 - phi_p) * R
```

[8]:
```
X2 = R*np.cos(phi_p) - l_p*np.sin(phi_p)
Y2 = R*np.sin(phi_p) + l_p*np.cos(phi_p)
np.shape(X2), np.shape(Y2)
```

[8]:
```
((1000,), (1000,))
```

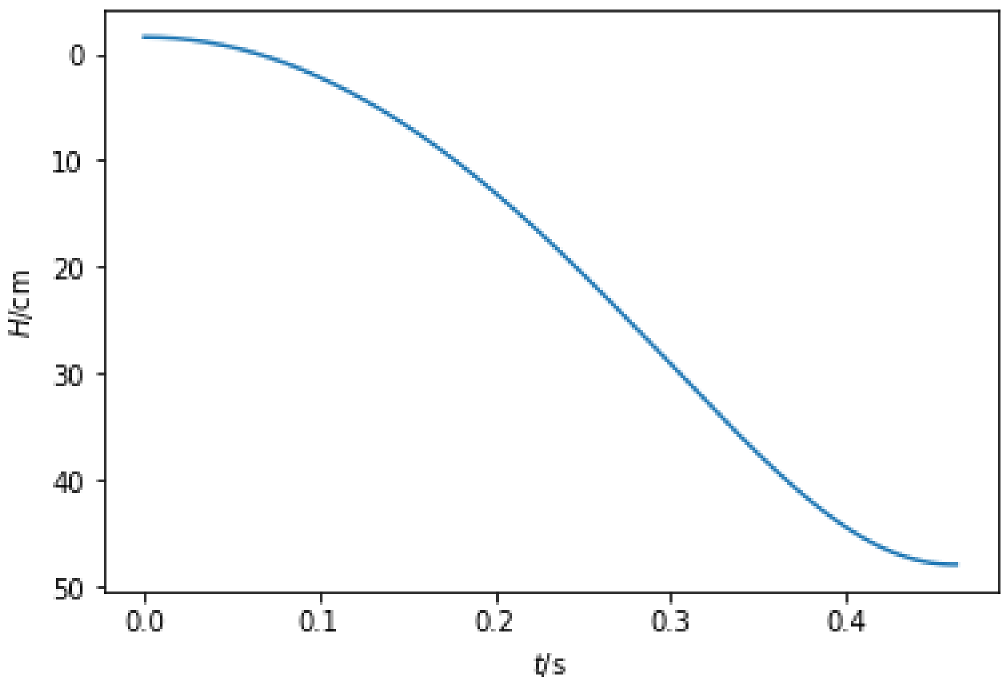
[9]:
```
#
%matplotlib inline
plt.plot(X1,Y1)
plt.plot(X2,Y2)
plt.xlabel("$X$/cm")
plt.ylabel("$Y$/cm")
plt.axis("equal")
plt.show()
```



[10]:
```
#                     –
H = l_0 - (U[:,0] + R * U[:,1])
np.shape(t), np.shape(H)                #
```

[10]:
```
((464,), (464,))
```

[11]:
```
%matplotlib inline
plt.plot(t,H)
plt.xlabel("$t$/s")
plt.ylabel("$H$/cm")
plt.gca().invert_yaxis()                #      y
```



4 求解第二阶段的运动细节

接下来我们希望得到第二阶段运动的细节，讨论绕绳的静摩擦力是否能够维持。对第二阶段后的运动进行研究。由于存在约束条件

$$l_0 - l = R(\varphi - \varphi_0)$$

建立自然坐标，对其进行运动分析。不难得出，在运动中

$$\mathrm{d}\varphi = \frac{\mathrm{d}l}{R} = \frac{|\mathrm{d}\vec{v}_\mathrm{n}|}{v} = \frac{v\mathrm{d}t}{l}$$

法向加速度即为

$$a_\mathrm{n} = \frac{v}{R}\frac{\mathrm{d}l}{\mathrm{d}t} = \frac{v^2}{l} = l\dot{\varphi}^2$$

而由于 $v = l\dot{\varphi}$，切向加速度

$$a_\text{ø} = \dot{v} = l\ddot{\varphi} + \dot{l}\dot{\varphi} = l\ddot{\varphi} - R\dot{\varphi}^2$$

故由此列出轻负载的第二阶段动力学方程

$$l\ddot{\varphi} = R\dot{\varphi}^2 + g\sin\varphi$$

$$F + mg\cos\varphi = ml\dot{\varphi}^2$$

若 $Fe^{\mu\varphi} \leq Mg$，则第二阶段状态失去稳定，运动将重回第一阶段。

为此，我们需要对程序作出较大的改动。首先将两个阶段分别定义成两个类。

[12]:
```
class Looping_Pendulum_Stage_1(object) :

    def __init__(self, m, M, u, R) :
        self.m, self.M, self.u, self.R = m, M, u, R

    def f(self, t, x) :
        l, phi, vl, vphi = x.tolist()

        Dl = vl
        Dphi = vphi

        A = np.mat([[self.m + self.M*np.exp(-self.u*phi)            ,   (self.m␣
 ↪+ self.M*np.exp(-self.u*phi))*self.R    ],
                    [(self.m + self.M)*self.R                       ,   self.
 ↪m*l**2 + (self.m + self.M)*self.R**2       ]])
        b = np.mat([[self.m*l*vphi**2 - self.M*g*np.exp(-self.u*phi) - self.
 ↪m*g*np.cos(phi)],
                    [-2*self.m*l*vl*vphi - self.m*g*(self.R*np.cos(phi)-l*np.
 ↪sin(phi)) - self.M*g*self.R ]])
        x = solve(A,b)

        Dvl, Dvphi = x[0], x[1]

        return [Dl, Dphi, Dvl, Dvphi]
```

[13]:
```
class Looping_Pendulum_Stage_2(object) :

    def __init__(self, l0, phi0, R) :
        self.l0, self.phi0, self.R = l0, phi0, R

    def f(self, t, x) :
        Phi, v = x.tolist()
        l = self.l0 - self.R * (Phi-self.phi0)
        DPhi = v
        Dv = (g*np.sin(Phi) + self.R*v**2) / l
        return [DPhi, Dv]
```

然后定义两个求解函数。这两个函数分别设有两种运动状态之间的转换接口，包括两个变量：1. `hold` 用以判断求解完成后重负载是否处于静止状态。2. `stop` 用以判断是否终止求解。

[14]:
```
def Solve_Stage_1(initials, parameters, t0, T=5.0) :
    """


    * initials -          list          4



    * parameters -       tuple          4

    """
    m, M, u, R = tuple(parameters)
    initials = list(initials)

    system = Looping_Pendulum_Stage_1(m=m, M=M, u=u, R=R)
    r = ode(system.f)
    r.set_initial_value(initials, t0)
    r.set_integrator("vode", method="bdf")
    dt = 1.0e-4 * R

    t = [t0,]
    U = [initials,]
    while r.successful() :
        if r.t+dt >= T or r.y[0] < 0 or r.y[2] > 0 :
            hold = False
            stop = True
            break
        if (r.y[2] + R*r.y[3]) > 0 :
            hold = True
            stop = False
            break
        r.integrate(r.t+dt)
        t.append(r.t)
        U.append(r.y)

    return np.array(U), np.array(t), hold, stop
```

[15]:

```
def Solve_Stage_2(initials, parameters, t0, T=5.0) :
    """


    * initials -                   2


    * parameters -                 2


    """
    phi0 = initials[0]
    l0, R = tuple(parameters)
    initials = list(initials)

    system = Looping_Pendulum_Stage_2(l0=l0, phi0=phi0, R=R)
    q = ode(system.f)
    q.set_initial_value(initials, t0)
    q.set_integrator("vode", method="bdf")
    dt = 1.0e-3 * R

    t = [t0,]
    U = [initials,]
    while q.successful() :
        if q.t+dt >= T or (l0 - R*(q.y[0]-phi0)) < 0 or q.y[1] > 100 :
            # q.y[1]<=100
            stop = True
            hold = True
            break
        F = - m*g*np.cos(q.y[0]) + m*(l0 - R*(q.y[0]-phi0))*q.y[1]**2
        if F < M*g*np.exp(-u*q.y[0]) :
            stop = False
            hold = False
            break
        q.integrate(q.t+dt)
        t.append(q.t)
        U.append(q.y)

    return np.array(U), np.array(t), hold, stop
```

随后需要修改求解过程，实际运动需要在第一、第二阶段之间相互转换，下面的程序加入了这一转换过程。

[16]:
```
hold = False
stop = False
initials = [100., pi/2, 0., 0.]
t0 = 0
U = []
while not stop :
    if hold :
        state = hold
        parameters = [initials[0], R]
        Us, ts, hold, stop = Solve_Stage_2([initials[1],initials[3]],␣
 ↪parameters, t0)
        l0, phi0 = initials[0], initials[1]
        Us = np.vstack((l0-R*(Us[:,0]-phi0), Us[:,0], -R*Us[:,1], Us[:,1])).
 ↪transpose()
        initials = Us[-1,:]
        t0 = ts[-1]
        U.append([ts, Us, state])
    else :
        state = hold
        Us, ts, hold, stop = Solve_Stage_1(initials, [m, M, u, R], t0)
        initials = Us[-1,:]
        t0 = ts[-1]
        U.append([ts, Us, state])
```

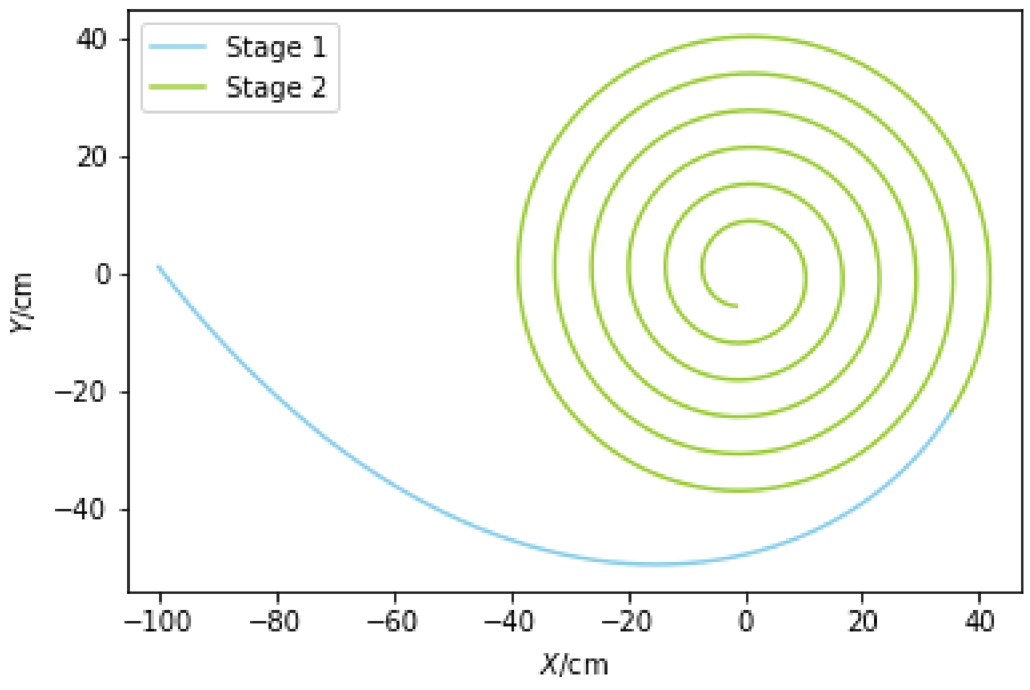
在上面的代码中，变量 U 是一个 list 变量，其每一个元素都是这样的一个 list 变量，它的第 0 个元素是时间轴，第 1 个元素是求解所得的原始数据，第 2 个元素是表示重负载是否处于静止悬挂状态的 boolean 变量。

下面我们对计算结果进行作图。首先是作轻负载的运动轨迹图。

[17]:
```
%matplotlib inline
plt.figure()
```

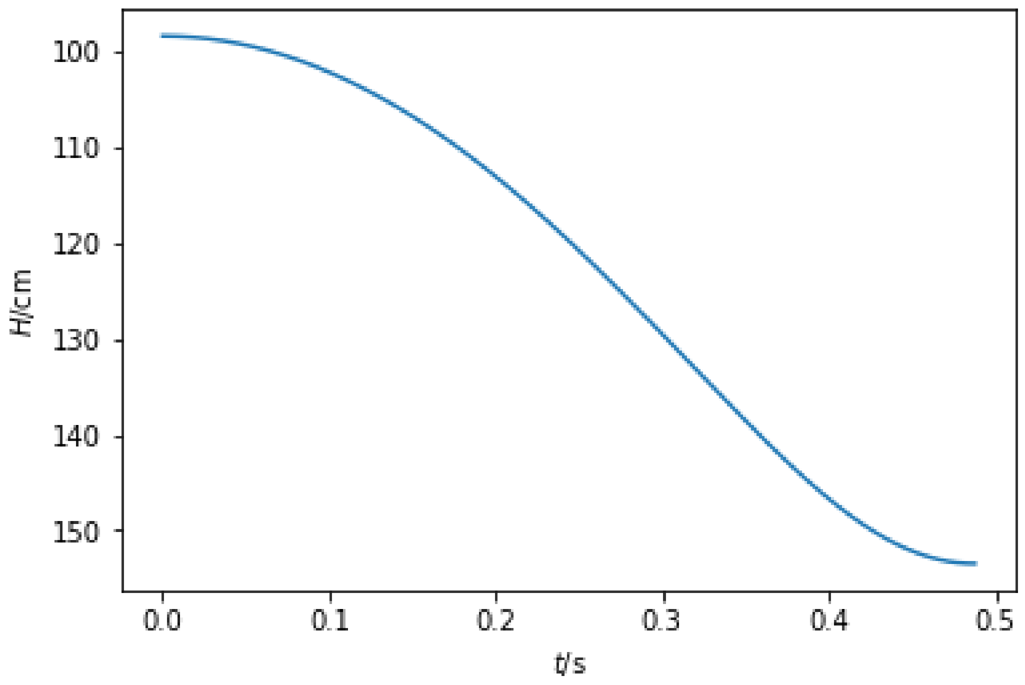
```
for Motion in U :
    L = Motion[1][:,0]
    Phi = Motion[1][:,1]
    X = R*np.cos(Phi) - L*np.sin(Phi)
    Y = R*np.sin(Phi) + L*np.cos(Phi)
    if Motion[2] :
        plt.plot(X, Y, color="yellowgreen")
    else:
        plt.plot(X, Y, color="skyblue")
plt.xlabel("$X$/cm")
plt.ylabel("$Y$/cm")
plt.axis("equal")
plt.legend(["Stage 1", "Stage 2"])
```

[17]:
```
<matplotlib.legend.Legend at 0x1521ed1978>
```



暂时还无法看出与先前求解的区别，这是因为当前条件下很难观察到「二段跳」之类的现象。然后作重负载下落的位置-时间图像。

[18]:
```
%matplotlib inline
S = 200.0
plt.figure()
for Motion in U :
    if not Motion[2] :
        t = Motion[0]
        H = S - (Motion[1][:,0] + R * Motion[1][:,1])
        plt.plot(t,H)
plt.xlabel("$t$/s")
plt.ylabel("$H$/cm")
plt.gca().invert_yaxis()                #       y
```



这幅图也与最初的解相一致。