三重振り子の解析

高山氏

2018 年 4 月 25 日

目次

1 三重振り子の運動方程式

この時，空気抵抗，支点の摩擦は無視する．

3 つの質点がたるまない糸によって繋がれていてある一点で支えられている，図 1 の状況を考える．そしてこの 3 つの質点がそれぞれどのような運動をするのか考える．

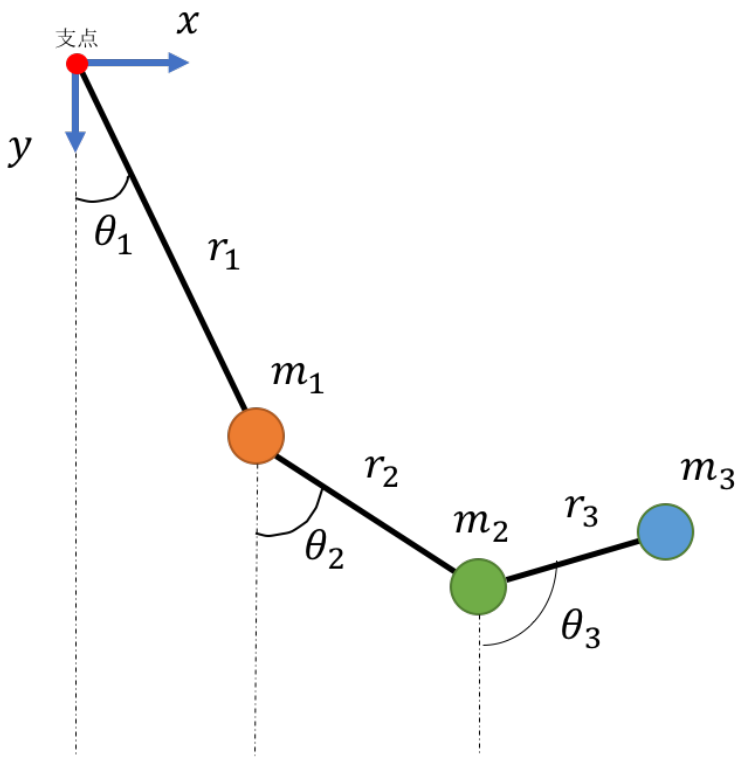

図 1　三重振り子の座標

この時，図 1 のように支点を 0 点として x 軸と y 軸をとる．そして，この座標軸を基準にして m_1 の質点の座標 (x_1, y_1)，m_2 の質点の座標 (x_2, y_2)，m_3 の質点の座標 (x_3, y_3) とする．この時，図 1 のように変数を設定すると以下の関係が成り立つ．

$$\begin{cases} x_1 = r_1 \sin\theta_1 \\ y_1 = r_1 \cos\theta_1 \end{cases} \tag{1}$$

$$\begin{cases} x_2 = r_1 \sin\theta_1 + r_2 \sin\theta_2 \\ y_2 = r_1 \cos\theta_1 + r_2 \cos\theta_2 \end{cases} \tag{2}$$

$$\begin{cases} x_3 = r_1 \sin\theta_1 + r_2 \sin\theta_2 + r_3 \sin\theta_3 \\ y_3 = r_1 \cos\theta_1 + r_2 \cos\theta_2 + r_3 \cos\theta_3 \end{cases} \tag{3}$$

この時の質点 1，2，3 の運動エネルギー，ポテンシャルを $T_1, U_1, T_2, U_2, T_3, U_3$ とするとそれぞれ以下のようになる．

$$\begin{cases} T_1 = \frac{1}{2} m_1 (r_1 \dot{\theta_1})^2 \\ U_1 = m_1 g r_1 \cos\theta_1 \end{cases} \tag{4}$$

$$\begin{cases} T_2 = \frac{1}{2}m_2[(r_1\dot{\theta_1})^2 + (r_2\dot{\theta_2})^2 + 2r_1r_2\dot{\theta_1}\dot{\theta_2}\cos(\theta_1 - \theta_2)] \\ U_2 = m_2g(r_1\cos\theta_1 + r_2\cos\theta_2) \end{cases} \tag{5}$$

$$\begin{cases} T_3 = \dfrac{m_3}{2}[(r_1\dot{\theta_1})^2 + (r_2\dot{\theta_2})^2 + (r_3\dot{\theta_3})^2 \\ \quad + 2r_1r_2\dot{\theta_1}\dot{\theta_2}\cos(\theta_1 - \theta_2) + 2r_2r_3\dot{\theta_2}\dot{\theta_3}\cos(\theta_2 - \theta_3) + 2r_3r_1\dot{\theta_3}\dot{\theta_1}\cos(\theta_3 - \theta_1)] \\ U_3 = m_3g(r_1\cos\theta_1 + r_2\cos\theta_2 + r_3\cos\theta_3) \end{cases} \tag{6}$$

一般に全系の運動エネルギー T は $T = \sum_{i=1}^{N} \frac{1}{2}m_i\left(\dot{x_i}^2 + \dot{y_i}^2\right)$ と表せる．また同様にポテンシャルも和で表され，この系のラグランジアン L は以下のようになる．

$$L = (T_1 + T_2 + T_3) - (U_1 + U_2 + U_3) \tag{7}$$

今回の場合だと r_1, r_2, r_3 は不変であるので，$\theta_1, \theta_2, \theta_3$ に関するオイラー・ラグランジュ方程式を立てれば良い．

$$\begin{cases} \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta_1}} = \frac{\partial L}{\partial \theta_1} \\ \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta_2}} = \frac{\partial L}{\partial \theta_2} \\ \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta_3}} = \frac{\partial L}{\partial \theta_3} \end{cases} \tag{8}$$

オイラー・ラグランジュ方程式に代入するために先にそれぞれ計算する．

$$\begin{aligned} \frac{\partial L}{\partial \dot{\theta_1}} &= (m_1 + m_2 + m_3)r_1^2\dot{\theta_1} + (m_2 + m_3)r_1r_2\dot{\theta_2}\cos(\theta_1 - \theta_2) \\ &\quad + (m_2 + m_3)r_3r_1\dot{\theta_3}\cos(\theta_3 - \theta_1) \\ \frac{\partial L}{\partial \theta_1} &= -(m_2 + m_3)r_1r_2\dot{\theta_1}\dot{\theta_2}\sin(\theta_1 - \theta_2) + (m_2 + m_3)r_3r_1\dot{\theta_3}\dot{\theta_1}\sin(\theta_3 - \theta_1) \\ &\quad + (m_1 + m_2 + m_3)gr_1\sin\theta_1 \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial \dot{\theta_2}} &= (m_2 + m_3)r_2^2\dot{\theta_2} + (m_2 + m_3)r_1r_2\dot{\theta_1}\cos(\theta_1 - \theta_2) \\ &\quad + m_3r_2r_3\dot{\theta_3}\cos(\theta_2 - \theta_3) \\ \frac{\partial L}{\partial \theta_2} &= (m_2 + m_3)r_1r_2\dot{\theta_1}\dot{\theta_2}\sin(\theta_1 - \theta_2) - m_3r_2r_3\dot{\theta_2}\dot{\theta_3}\sin(\theta_2 - \theta_3) \\ &\quad + (m_2 + m_3)gr_2\sin\theta_2 \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial \dot{\theta_3}} &= m_3r_3^2\dot{\theta_3} + m_3r_3r_1\dot{\theta_1}\cos(\theta_3 - \theta_1) + m_3r_2r_3\dot{\theta_2}\cos(\theta_2 - \theta_3) \\ \frac{\partial L}{\partial \theta_3} &= -m_3r_3r_1\dot{\theta_3}\dot{\theta_1}\sin(\theta_3 - \theta_1) + m_3r_2r_3\dot{\theta_2}\dot{\theta_3}\sin(\theta_2 - \theta_3) \\ &\quad + m_3gr_3\sin\theta_3 \end{aligned}$$

こっからさらに時間微分してごちゃごちゃまとめる．（気合いって大事ですね）

$$
\begin{aligned}
&\begin{pmatrix} 1 & \frac{m_2+m_3}{m_1+m_2+m_3}\frac{r_2}{r_1}\cos(\theta_1-\theta_2) & \frac{m_3}{m_1+m_2+m_3}\frac{r_3}{r_l}\cos(\theta_1-\theta_3) \\ \frac{r_1}{r_2}\cos(\theta_2-\theta_1) & 1 & \frac{m_3}{m_2+m_3}\frac{r_3}{r_2}\cos(\theta_2-\theta_3) \\ \frac{r_1}{r_3}\cos(\theta_3-\theta_1) & \frac{r_2}{r_3}\cos(\theta_3-\theta_2) & 1 \end{pmatrix}\begin{pmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\\ \ddot{\theta}_3\end{pmatrix} \\
=&\begin{pmatrix} 0 & -\frac{m_2+m_3}{m_1+m_2+m_3}\frac{r_2}{r_1}\sin(\theta_1-\theta_2) & -\frac{m_3}{m_1+m_2+m_3}\frac{r_3}{r_l}\sin(\theta_1-\theta_3) \\ -\frac{r_1}{r_2}\sin(\theta_2-\theta_1) & 0 & -\frac{m_3}{m_2+m_3}\frac{r_3}{r_2}\sin(\theta_2-\theta_3) \\ -\frac{r_1}{r_3}\sin(\theta_3-\theta_1) & -\frac{r_2}{r_3}\sin(\theta_3-\theta_2) & 0 \end{pmatrix}\begin{pmatrix}(\dot{\theta}_1)^2\\ (\dot{\theta}_2)^2\\ (\dot{\theta}_3)^2\end{pmatrix} \\
&+g\begin{pmatrix}\frac{\sin\theta_1}{r_1}\\ \frac{\sin\theta_2}{r_2}\\ \frac{\sin\theta_3}{r_3}\end{pmatrix}
\end{aligned}
$$

これを解いてくり～＾

2　数値計算

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

# まずどれくらいの時間回すか
# 秒数
T = 20.
# ステップ数
N = 4000
# 間隔
dt = T/float(N)
# 時間の配列を作る
t = np.arange(0.0, T, T/float(N))
count = -1
b = 1. / 6.

# おもりに関する初期条件
r1 = 2.
r2 = 1.
r3 = 1.

m1 = 2.
m2 = 1.
m3 = 1.
M = m1 + m2 + m3

g = 9.8

# おもりの位置に関する初期条件
theta1_0 = (np.pi / 180.) * 60.
theta2_0 = (np.pi / 180.) * 0.
theta3_0 = (np.pi / 180.) * 0.

# 角度をN行3列の配列に格納する（各行に角度の情報を入れていく）
theta = np.empty([N, 3])
```

```
# 初めの角度を初めの行に入れる
theta[0] = np.array([theta1_0, theta2_0, theta3_0])
# thetaの一階微分の配列を作る
omega = np.empty([N, 3])
omega[0] = np.array([0., 0., 0.])


# 二次元直交座標に直す用の配列
x = np.empty([N, 3])
y = np.empty([N, 3])
E = np.empty(N)


# r, thetaから直交座標に変換するような関数を作る.
def carx(r, angle):
    return r * np.sin(angle)


def cary(r, angle):
    return r * np.cos(angle)


# さっき作った関数を利用してthetaを入れるだけで3つの質点の座標を作ってもらう
def carxx(angle):
    xx = np.array([0., 0., 0.])
    xx[0] = carx(r1, angle[0])
    xx[1] = carx(r1, angle[0]) + carx(r2, angle[1])
    xx[2] = carx(r1, angle[0]) + carx(r2, angle[1]) + carx(r3, angle[2])
    return xx


def caryy(angle):
    yy = np.array([0., 0., 0.])
    yy[0] = cary(r1, angle[0])
    yy[1] = cary(r1, angle[0]) + cary(r2, angle[1])
    yy[2] = cary(r1, angle[0]) + cary(r2, angle[1]) + cary(r3, angle[2])
    return yy


# 左辺の奴にかかってる行列を作る
def I1(angle):
    I = np.array([[1., (m2 + m3) * np.cos(angle[0] - angle[1]) * r2 / (r1 * M),
                   m3 * np.cos(angle[0] - angle[2]) * r3 / (r1 * M)],
                  [r1 * np.cos(angle[1] - angle[0]) / r2, 1.,
                   m3 * np.cos(angle[1] - angle[2]) * r3 / (r2 * (m2 + m3))],
                  [r1 * np.cos(angle[2] - angle[0]) / r3,
                   r2 * np.cos(angle[2] - angle[1]) / r3, 1.]])
    return I


# 右辺第一項の行列
def I2(angle):
    I = np.array([[0., -(m2 + m3) * np.sin(angle[0] - angle[1]) * r2 / (r1 * M),
                   -m3 * np.sin(angle[0] - angle[2]) * r3 / (r1 * M)],
                  [-r1 * np.sin(angle[1] - angle[0]) / r2, 0.,
                   -m3 * np.sin(angle[1] - angle[2]) * r3 / (r2 * (m2 + m3))],
                  [-r1 * np.sin(angle[2] - angle[0]) / r3,
```

```
                    -r2 * np.sin(angle[2] - angle[1]) / r3, 0.]])
    return I


# ポテンシャルの行列
def I3(angle):
    I = np.array([[g * np.sin(angle[0]) / r1],
                  [g * np.sin(angle[1]) / r2],
                  [g * np.sin(angle[2]) / r3]])
    return I


# thetaの二回微分に関する微分方程式を返す
def oed(angle, dangle):
    # 左辺の行列の逆行列を作る. 1行でできる. すごい.
    Iinv = np.linalg.inv(I1(angle))
    # 行列の掛け算です
    Iv = np.dot(Iinv, I2(angle))
    # thetaの一階微分の2乗の行列
    dd = np.array([[dangle[0] ** 2],
                   [dangle[1] ** 2],
                   [dangle[2] ** 2]])
    Ivd = np.dot(Iv, dd)
    U = np.dot(Iinv, I3(angle))
    return Ivd + U


# 最後ルンゲクッタを書きやすくするために定義
def runge_kutta(a, kg1, kg2, kg3, kg4):
    k1 = kg1 * np.array([b * dt])
    k2 = kg2 * np.array([2. * b * dt])
    k3 = kg3 * np.array([2. * b * dt])
    k4 = kg4 * np.array([b * dt])
    return a + k1 + k2 + k3 + k4

# thetaとomegaを入れると力学的エネルギーを返す関数
def Um(theta, omega):
    T1 = m1 * (r1 * omega[0]) ** 2 / 2.
    T2 = m2 * (
    (r1 * omega[0]) ** 2 + (r2 * omega[1]) ** 2 + 2. * r1 * r2 * omega[0] * omega[1] * np.cos(theta[0]
    T3 = m3 * ((r1 * omega[0]) ** 2 + (r2 * omega[1]) ** 2 + (r3 * omega[2]) ** 2
               + 2. * r1 * r2 * omega[0] * omega[1] * np.cos(theta[0] - theta[1])
               + 2. * r2 * r3 * omega[1] * omega[2] * np.cos(theta[1] - theta[2])
               + 2. * r3 * r1 * omega[2] * omega[0] * np.cos(theta[2] - theta[0])) /2.
    U1 = m1 * g * r1 * np.cos(theta[0])
    U2 = m2 * g * (r1 * np.cos(theta[0]) + r2 * np.cos(theta[1]))
    U3 = m3 * g * (r1 * np.cos(theta[0]) + r2 * np.cos(theta[1]) + r3 * np.cos(theta[2]))
    return T1 + T2 + T3 + U1 + U2 + U3


E[0] = Um(theta[0], omega[0])

for i in range(N - 1):
    t[i + 1] = t[i] + dt
    # ルンゲクッタで解く
    ko1 = oed(theta[i], omega[i])
    kt1 = omega[i]
```

```
    ko2 = oed(theta[i] + kt1 * np.array(dt / 2.),
              omega[i] + ko1.ravel() * np.array(dt / 2.))
    kt2 = omega[i] + ko1.ravel() * np.array(dt / 2.)

    ko3 = oed(theta[i] + kt2 * np.array(dt / 2.),
              omega[i] + ko2.ravel() * np.array(dt / 2.))
    kt3 = omega[i] + ko2.ravel() * np.array(dt / 2.)

    ko4 = oed(theta[i] + kt3 * np.array(dt),
              omega[i] + ko3.ravel() * np.array(dt))
    kt4 = omega[i] + ko3.ravel() * np.array(dt)

    omega[i + 1] = runge_kutta(omega[i], ko1.ravel(),
                               ko2.ravel(), ko3.ravel(), ko4.ravel())
    theta[i + 1] = runge_kutta(theta[i], kt1, kt2, kt3, kt4)

    # x, y座標に変換
    x[i + 1] = carxx(theta[i + 1])
    y[i + 1] = caryy(theta[i + 1])
    E[i + 1] = Um(theta[i], omega[i])
    print(E[i+1])

plt.title("Energy")
plt.plot(t, E)
plt.grid()
plt.show()

# 質点1, 2, 3の座標に分ける
x1 = x[:, 0]
x2 = x[:, 1]
x3 = x[:, 2]

y1 = y[:, 0]
y2 = y[:, 1]
y3 = y[:, 2]

# こっからアニメーション
fig = plt.figure()
ax = fig.add_subplot(111, autoscale_on=False, xlim=(-5., 5.), ylim=(-5., 5.))
ax.grid()

line, = ax.plot([], [], 'o-', lw=3)
time_template = 'time = %.1fs'
time_text = ax.text(0.005, 0.9, '', transform=ax.transAxes)


def init():
    line.set_data([], [])
    time_text.set_text('')
    return line, time_text


def animate(i):
    thisx = [0, x1[i], x2[i], x3[i]]
    thisy = [0, y1[i], y2[i], y3[i]]
```

```
    line.set_data(thisx, thisy)
    time_text.set_text(time_template % (i * dt))
    return line, time_text


ani = animation.FuncAnimation(fig, animate, np.arange(1, len(y)),
                              interval=2.5, blit=False, init_func=init)

ani.save('triple_pendulum.mp4', fps=15)
plt.show()
```

2.1 数値計算結果

この系の計算結果が本当に現実に即しているのか，系全体の力学的エネルギーを計算することで考える．図 2 に数値計算上での，各時刻での力学的エネルギーを示す．

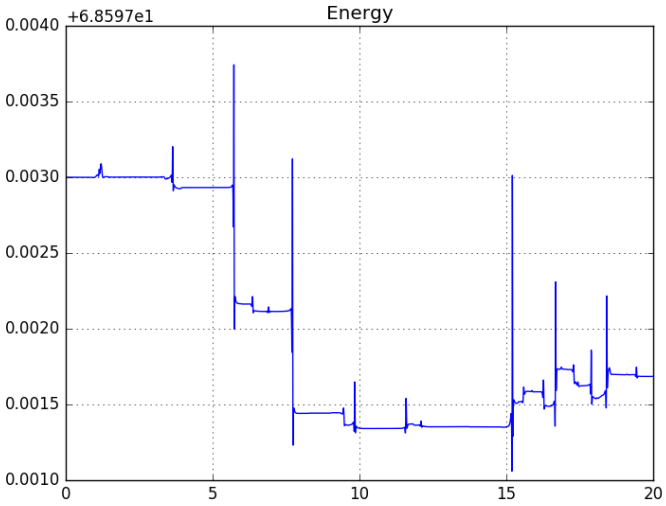

図 2　各時刻での力学的エネルギー

ちょっと見にくいんですけど，このグラフは 68.597 の値とのズレを表す．（グラフの左上にある値）まぁ，つまり t=0 の値からエネルギーは最大でも 0.002 程度しか変わっていないということをグラフは表している．これはエネルギー保存則をちゃんと表している．つまり，この数値計算はそれっぽい解を返してくれているはず．見てると周期性みたいなのが見えてきそうで面白い．

3　参考

3.1　オイラー・ラグランジュ方程式

ラグランジアン L として，最小作用の原理より以下のオイラー・ラグランジュ方程式が成り立つ．

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} = \frac{\partial L}{\partial q_i} \tag{9}$$

参考文献

[1] 三重振り子の振る舞い http://www.math.ryukoku.ac.jp/~tsutomu/undergraduate/2014/14_hayakawa_pa.pdf

[2] python で二重振り子の運動方程式 https://qiita.com/neka-nat@github/items/e4f13bb95b7c153a75e2