数値解析 最終レポート
単振り子および二重振り子の運動方程式に対する Runge-Kutta 法の適用

計数工学科 J4190147 小野悠太

2021 年 2 月 10 日

目次

1 目的 3

2 単振り子の運動 4
2.1 単振り子の運動方程式 4
2.2 単振り子の運動方程式に対する Runge-Kutta 法の適用 4
2.3 結果 6
2.3.1 単振り子の運動のアニメーション 6
2.3.2 エネルギー誤差の時間変化 6
2.3.3 時間幅の取り方による誤差の変化 8
2.4 考察 12

3 二重振り子の運動 13
3.1 二重振り子の運動方程式 13
3.2 二重振り子の運動方程式に対する Runge-Kutta 法の適用 13
3.2.1 Runge-Kutta 法の式について 13
3.2.2 Runge-Kutta 法の妥当性について 13
3.3 結果 13
3.3.1 二重振り子の運動のアニメーション 13
3.3.2 エネルギー誤差の時間変化 13
3.3.3 時間幅の取り方による誤差の変化 13
3.4 考察 14

4 付録 15
4.1 二重振り子の運動方程式の導出 15
4.2 二重振り子の微小角近似における解析解の導出 16
4.3 ソースコード 17
4.4 計算機環境 26
4.5 参考文献 26

1　目的

このレポートでは，二重振り子の運動方程式に対して Runge-Kutta 法を適用し数値解を求め，運動を可視化したうえで系の全エネルギーがどれほどの精度で保存されるのかを確かめることを目的とする．

このような目的を定めた理由としては，1A セメスターで受講した「振動波動論」で二重振り子のカオスが紹介されておりその複雑な運動に興味を持ったことや今セメスターの「数学 1D」で解析力学を習い，二重振り子の運動方程式を立てやすくなったことが挙げられる．また，カオスという複雑で非周期的な運動に対し Runge-Kutta 法を適用しても，期待される精度が出るのか気になったことも理由の一つである．

具体的な設定や実験方法については後の章で説明する．また，ここで扱う二重振り子のモデルはそれぞれの振り子の腕の先端に質点が存在するモデル（単振り子を連結したもの）とする．

2　単振り子の運動

この章では二重振り子ではなく単振り子を扱う．この運動はカオスではない．まず単振り子に対して Runge-Kutta 法を適用して系のエネルギーの変化を求めることで，カオスである場合とそうでない場合の比較を可能とすることを目的としている．
また，今回扱う二重振り子のモデルは単振り子を 2 つ直列に連結したものであるから，これらの結果に類似性が見られることも期待される．

2.1　単振り子の運動方程式

ここでは，振り子の腕の一方が原点 O に固定されており，他方の端点には質量 m_1 の質点が取り付けられているとする．
θ は鉛直下方向と振り子の腕がなす角とし，腕の長さを l とする．これは極めて一般的な振り子であり，その運動方程式は以下のようにあらわされる．

$$\ddot{\theta} = -\frac{g}{l}\sin\theta \tag{1}$$

$\dot{\theta} = \omega$ とするとこの運動方程式は，一階の微分方程式を連立させたものとなる．

$$\begin{cases} \dot{\theta} = \omega & \text{(2a)} \\ \dot{\omega} = -\dfrac{g}{l}\sin\theta & \text{(2b)} \end{cases}$$

2.2　単振り子の運動方程式に対する Runge-Kutta 法の適用

以下では式 (2) に対して Runge-Kutta 法を適用することを考える．

$$f(\theta) = -\frac{g}{l}\sin\theta \tag{3}$$

として，

$$k_1 = f(\theta^{(m)}) \tag{4}$$
$$n_1 = \omega^{(m)} \tag{5}$$
$$k_2 = f\left(\theta^{(m)} + \frac{n_1}{2}\Delta t\right) \tag{6}$$
$$n_2 = \omega^{(m)} + \frac{k_1}{2}\Delta t \tag{7}$$
$$k_3 = f\left(\theta^{(m)} + \frac{n_2}{2}\Delta t\right) \tag{8}$$
$$n_3 = \omega^{(m)} + \frac{k_2}{2}\Delta t \tag{9}$$
$$k_4 = f\left(\theta^{(m)} + n_3\Delta t\right) \tag{10}$$
$$n_4 = \omega^{(m)} + k_3\Delta t \tag{11}$$

と定義すると，

$$\omega^{(m+1)} = \omega^{(m)} + \left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right) \tag{12}$$

$$\theta^{(m+1)} = \theta^{(m)} + \left(\frac{1}{6}n_1 + \frac{1}{3}n_2 + \frac{1}{3}n_3 + \frac{1}{6}n_4\right) \tag{13}$$

と表せる．

今回はこれを Python により実装し，運動の可視化およびそのエネルギー変化を可視化した．運動の可視化のプログラムは付録中のソースコード 1，時間幅の取り方によるエネルギーの誤差の変化を可視化するプログラムはソースコード 2，エネルギーの時間変化のグラフを作成するプログラムはソースコード 3 に示す．

2.3 結果

2.3.1 単振り子の運動のアニメーション

ソースコード 1 では長さ l，質量 m，初角度 θ_0，初角速度 ω_0 がパラメータとして設定できるようになっているが，ここでは，$l = 1$, $m = 1$, $\omega_0 = 0$ として運動の様子を見る．
このプログラムでは運動の様子をシミュレーションしたアニメーションが gif ファイルとして保存できるが，このレポート上に gif を直接掲載するのは難しかったため，以下のリンク先に保存した．

- 初角 $\dfrac{\pi}{4}$ の単振り子
- 初角 $\dfrac{\pi}{3}$ の単振り子

これらのアニメーションでは特に不自然に見える箇所はなく，Runge-Kutta 法によって作成した単振り子の運動と人間が想像する単振り子の運動との間には大きな乖離はないと考えられる．

2.3.2 エネルギー誤差の時間変化

視覚的な運動が想像と一致していても，系としての構造をきちんと保っているとは限らない．
よって，ここではこの系に対して Runge-Kutta 法を適用したときにエネルギーが時間経過とともにどのように変化しているかを観察する．
本来この系では運動エネルギーとポテンシャルエネルギーの和（力学的エネルギー）が保存されており，そのような結果が得られることが望ましい．下の図 1 はソースコード 3 において初角度を $\theta_0 = \dfrac{\pi}{4}$ としたものであり，縦軸に真の力学的エネルギーと計算された力学的エネルギーの差 $(E_0 - E)$，横軸に経過時間をとっている．初期条件より E_0 は負の定数であるから，このグラフからは $E(< 0)$ が時間経過とともに小さくなっていっていることが読み取れる．
なお，図中の点線は $f(t) = 3 \times 10^{-9} t$ の直線である．

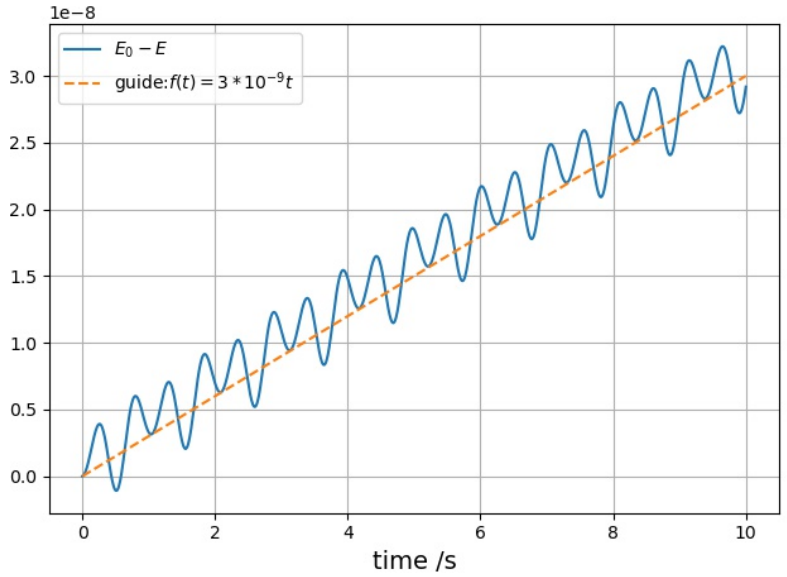

図 1　系のエネルギー誤差の時間変化

2.3.3　時間幅の取り方による誤差の変化

今回用いている Runge-Kutta 法は 4 次の精度を持つということが知られており，繰り返しの時間幅 Δt を $\dfrac{1}{10}$ 倍すれば誤差は $\dfrac{1}{10000}$ 倍になるはずである．
したがって，この節では時間幅の取り方によって誤差がどのように変化しているかを確認する．

以下に示す図 2 から図 5 はソースコード 2 において，初角度をそれぞれ $\dfrac{\pi}{2}, \dfrac{\pi}{3}, \dfrac{\pi}{4}, \dfrac{\pi}{6}$ と設定して得られたグラフである．なお，ここでの誤差 (Error) は 10 秒経過後の $|E - E_0|$ と定義した．

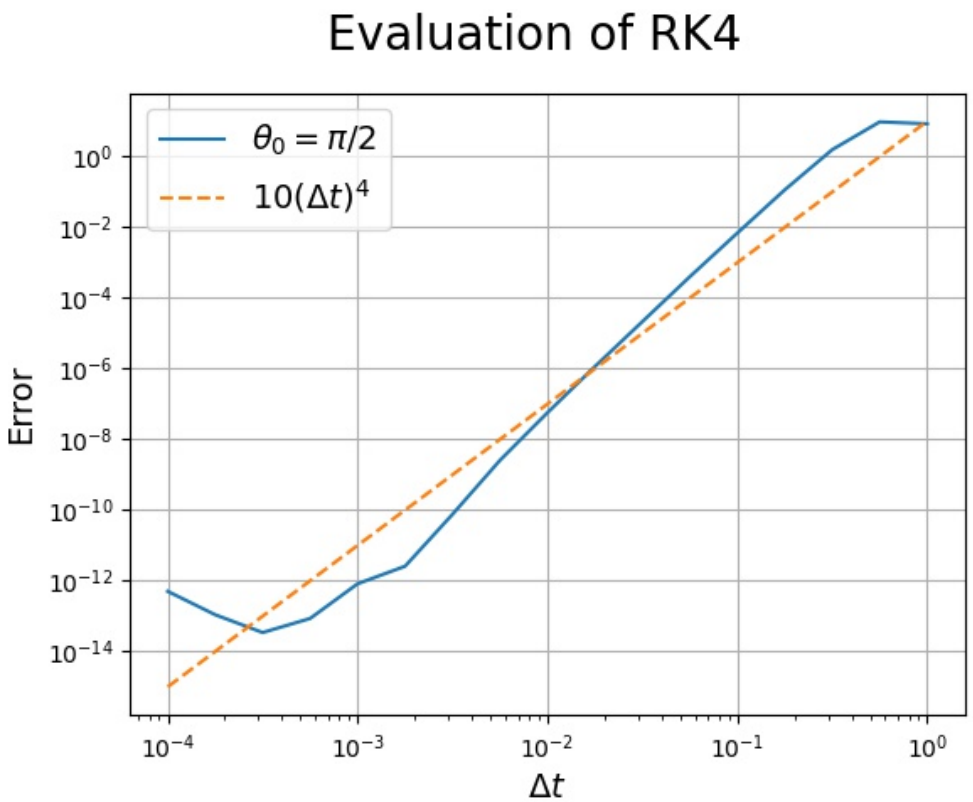

図 2　初角度 $\dfrac{\pi}{2}$における力学的エネルギーの精度

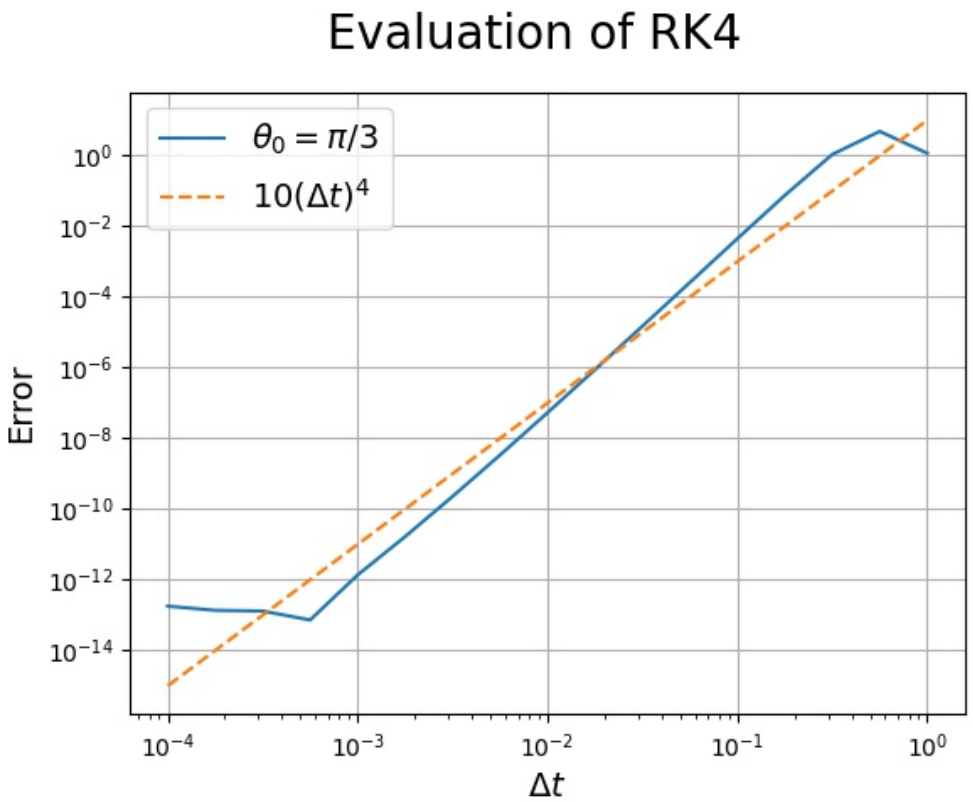

図 3　初角度 $\frac{\pi}{3}$における力学的エネルギーの精度

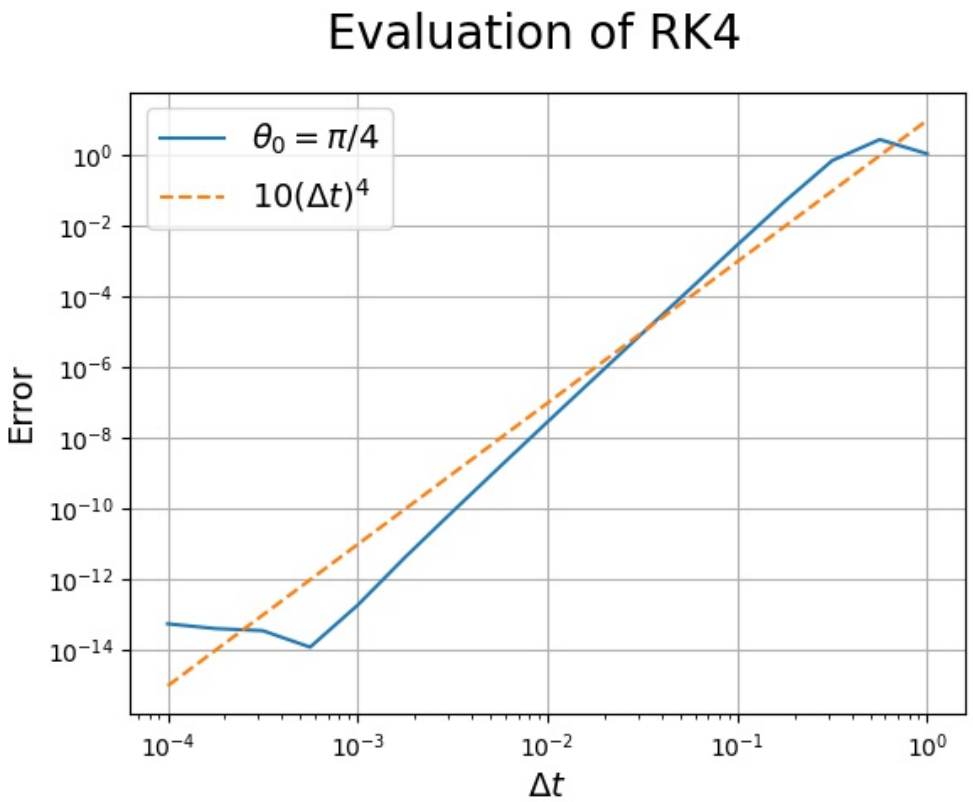

図 4　初角度 $\frac{\pi}{4}$における力学的エネルギーの精度

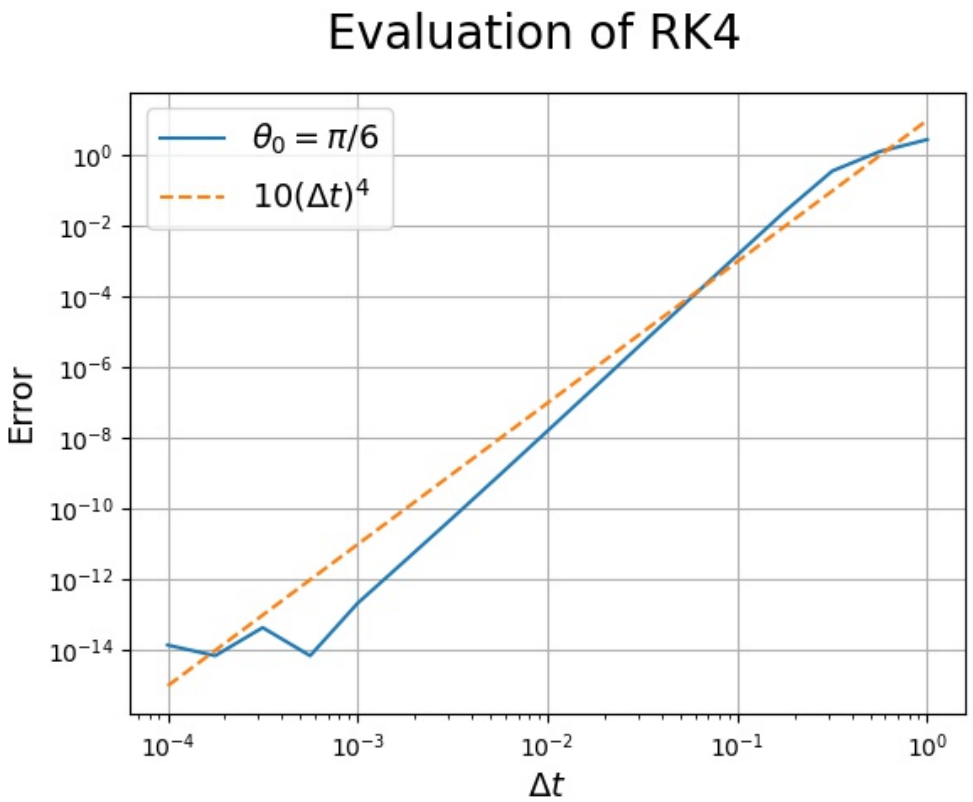

図 5　初角度 $\frac{\pi}{6}$ における力学的エネルギーの精度

2.4　考察

まずアニメーションから読みとることのできる視覚的な情報についてであるが，ここからは運動のシミュレーションに不自然な点が特にないという事しか読み取ることができない．一方で振動の減衰などがはっきりと確認できるわけではないことから，系の力学的エネルギーがあまり変化していないのではないかという予想が立てられる．

次に，エネルギー誤差の時間変化についてである．

この図にひかれている点線は $f(t) = 3 \times 10^{-9} t$ の補助線である．誤差は大域的にはこの直線に沿って増加し，局所的には振動している．ここに掲載した図 1 については初角度を $\theta_0 = \dfrac{\pi}{4}$ としたものであるが，他の初角度に対しても同じような傾向（大域的には誤差が増加，局所的には振動）が確認できた．振動しているように見える箇所は周期的に同じ形の変動を繰り返しているように見える．これは，それぞれの θ について計算におけるエネルギーの減少しやすさや増加しやすさがある程度決まっており，そこを「周期的」に通るため同じような振動を繰り返しているように見えるのではないかと考えた．またそのように考えた場合，大域的に減少しているのは，「1 周期」分の合計をとった時にその和が負の値であることによると考えられる（厳密な意味での周期的な運動ではないと考えられるため，「周期的」とあらわした）．

最後に時間幅の変化に対する誤差の変化についてである．図 2 から図 5 の図中に記された点線は $f(t) = 10(\Delta t)^4$ の補助線であり，4 次の精度が達成されているかの目安となる．これらのグラフは両対数グラフになっていることに注意すると，この補助線よりも傾きが大きければ 4 次の精度を持っていると考えてよいと言える．

図より，どの初角度に対しても $\Delta t = 10^0$ から $\Delta t = 10^{-3}$ あたりにかけては 4 次の精度が成り立っている．一方で，Δt が 10^{-3} よりも小さくなると誤差が増加傾向に転じる．これについて明確な理由はわかっていないが，上で述べたように，θ に対応して誤差の変化のしやすさが決まっていると仮定し，θ が大きい点（つまり角速度 ω が小さい点）における計算で誤差が広がりやすい場合を考えると，時間幅 Δt が小さくなるにつれて θ が大きい点で計算が実行される回数が増えていき，結果として一周期分の誤差増大量が大きくなると考えることができ辻褄が合う．

この仮定があっているかは，「1 周期」分の計算においてどのような θ で誤差が大きくなっているのかを確認すれば確かめられると考えたが，今回は時間不足で実施できなかった．

3　二重振り子の運動

3.1　二重振り子の運動方程式

3.2　二重振り子の運動方程式に対する Runge-Kutta 法の適用

3.2.1　Runge-Kutta 法の式について

3.2.2　Runge-Kutta 法の妥当性について

二重振り子の運動は θ_1, θ_2 が十分に小さいときのみ，近似を用いて解析解を求めることができる．したがって，微小角における解析解と数値解を比較することで，前節で定めた Runge-Kutta 法の式による計算が妥当であるのかを確認することができる．

ここでは，$\theta_1, \theta_2 << 1$ である状況として，$\theta_1 = \theta_2 = 0.001, \dot{\theta_1} = \dot{\theta_2} = 0$ と設定する．

3.3　結果

3.3.1　二重振り子の運動のアニメーション

3.3.2　エネルギー誤差の時間変化

3.3.3　時間幅の取り方による誤差の変化

3.4　考察

4　付録

4.1　二重振り子の運動方程式の導出

4.2　二重振り子の微小角近似における解析解の導出

4.3　ソースコード

ソースコード 1　単振り子のアニメーション作成

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as anim
from collections import deque
from datetime import datetime


# constants
g = 9.80665 # Standard gravity
dt_now = datetime.now()


# parameters
l1 = 1
m1 = 1
theta1 = np.pi/3
w1 = 0
t_start = 0
t_end = 10
steps = 1000


# calculated
dt = (t_end - t_start)/steps


# lists
theta_series = deque([theta1])
w_series = deque([w1])
xs = deque([l1*np.sin(theta1)])
ys = deque([-l1*np.cos(theta1)])
t_series = np.linspace(t_start, t_end, steps+1)
T_series = deque([m1*(l1**2)*(w1**2)/2])
U_series = deque([-m1*g*l1*np.cos(theta1)])
H_series = deque([m1*(l1**2)*(w1**2)/2-m1*g*l1*np.cos(theta1)])


fig, ax = plt.subplots()
```

```
ax.set_xlabel(R'$x\␣/m$', fontsize=14)
ax.set_ylabel(R'$y\␣/m$', fontsize=14)


def f(theta):
    return -g*np.sin(theta)/l1


def RungeKutta41(theta, w):
    k1 = f(theta) # w
    n1 = w # theta
    k2 = f(theta + n1*dt/2)
    n2 = w + k1*dt/2
    k3 = f(theta + n2*dt/2)
    n3 = w + k2*dt/2
    k4 = f(theta + n3*dt)
    n4 = w + k3*dt
    w = w + (k1/6 + k2/3 + k3/3 + k4/6)*dt
    theta = theta + (n1/6 + n2/3 + n3/3 + n4/6)*dt

    w_series.append(w)
    theta_series.append(theta)
    xs.append(l1*np.sin(theta))
    ys.append(-l1*np.cos(theta))
    T = m1*(l1**2)*(w**2)/2
    U = - m1*g*l1*np.cos(theta)
    H = T + U
    T_series.append(T)
    U_series.append(U)
    H_series.append(H)


for i in range(steps):
    theta = theta_series[-1]
    w = w_series[-1]
    RungeKutta41(theta, w)

images = []

for i in range(steps):
    x = [0, xs[i]]
    y = [0, ys[i]]
```

```
81      image = ax.plot(x, y, 'o-', lw=2, c="black", label="pendulum")
82      ax.grid(True)
83      ax.axis('equal')
84      ax.set_title("simple␣pendulum", fontsize=18)
85      images.append(image)
86
87  ani = anim.ArtistAnimation(fig, images, interval=10)
88  plt.tight_layout()
89
90  ani.save('./figure/RK41/animation/{0}-{1}-{2}-{3}{4}{5}.gif'.format(
91      dt_now.year, dt_now.month, dt_now.day,
92      dt_now.hour, dt_now.minute, dt_now.second), writer='pillow', fps=50)
93  # plt.show()
```

ソースコード 2　単振り子における RK4 の次数について

```
import numpy as np
import matplotlib.pyplot as plt
from collections import deque
from datetime import datetime


# constants
g = 9.80665 # Standard gravity
dt_now = datetime.now()

# parameters
l1 = 1
m1 = 1
w0 = 0
t_start = 0
t_end = 10

# lists
T_series = deque([])
U_series = deque([])
H_series = deque([])


def f(theta):
    return -g*np.sin(theta)/l1


def RungeKutta41(theta, w, dt):
    k1 = f(theta) # w
    n1 = w # theta
    k2 = f(theta + n1*dt/2)
    n2 = w + k1*dt/2
    k3 = f(theta + n2*dt/2)
    n3 = w + k2*dt/2
    k4 = f(theta + n3*dt)
    n4 = w + k3*dt
    w = w + (k1/6 + k2/3 + k3/3 + k4/6)*dt
    theta = theta + (n1/6 + n2/3 + n3/3 + n4/6)*dt

    w_series.append(w)
    theta_series.append(theta)
```

```
    T = m1*(l1**2)*(w**2)/2
    U = - m1*g*l1*np.cos(theta)
    H = T + U
    T_series.append(T)
    U_series.append(U)
    H_series.append(H)


thetas = [np.pi/6, np.pi/4, np.pi/3, np.pi/2]
thetas_str = ["pi6", "pi4", "pi3", "pi2"]
thetas_tex = [R"\pi/6", R"\pi/4", R"\pi/3", R"\pi/2"]
dts = np.logspace(-4, 0, num=17, base=10.0)


def guide(t):
    return 10*t**4


for theta0, theta0_str, theta0_tex in zip(thetas, thetas_str, thetas_tex):
    fig, ax = plt.subplots()
    exact_H = m1*(l1**2)*(w0**2)/2 - m1*g*l1*np.cos(theta0)
    y = []
    for dt in dts:
        steps = int((t_end - t_start) / dt)
        theta_series = deque([theta0])
        w_series = deque([w0])
        for i in range(steps):
            theta = theta_series[-1]
            w = w_series[-1]
            RungeKutta41(theta, w, dt)
        y.append(np.abs(exact_H - H_series[-1]))
    ax.plot(dts, y, label=R"$\theta_0␣={}$".format(theta0_tex))
    ax.plot(dts, guide(dts), label=R"$10(\Delta␣t)^4$", ls="dashed")
    ax.semilogx()
    ax.semilogy()
    ax.grid()
    ax.legend(fontsize="14")
    ax.set_xlabel(R"$␣\Delta␣t$", fontsize="14")
    ax.set_ylabel("Error", fontsize="14")
    fig.suptitle('Evaluation␣of␣RK4', fontsize="20")
    fig.savefig('./figure/RK41/evaluation1/{6}_{0}-{1}-{2}-{3}{4}{5}.jpeg'
                .format(dt_now.year, dt_now.month, dt_now.day, dt_now.hour,
```

```
84                          dt_now.minute, dt_now.second, theta0_str))
```

ソースコード 3　単振り子のエネルギーの時間変化について

```
import numpy as np
import matplotlib.pyplot as plt
from collections import deque
from datetime import datetime

# constants
g = 9.80665 # Standard gravity
dt_now = datetime.now()

# parameters
l1 = 1
m1 = 1
theta1 = np.pi/4
w1 = 0
t_start = 0
t_end = 10
steps = 1000

# calculated
dt = (t_end - t_start)/steps

# lists
theta_series = deque([theta1])
w_series = deque([w1])
xs = deque([l1*np.sin(theta1)])
ys = deque([-l1*np.cos(theta1)])
t_series = np.linspace(t_start, t_end, steps+1)
T_series = deque([m1*(l1**2)*(w1**2)/2])
U_series = deque([-m1*g*l1*np.cos(theta1)])
H_series = deque([m1*(l1**2)*(w1**2)/2-m1*g*l1*np.cos(theta1)])

fig = plt.figure()
ax = fig.add_subplot(111)


def f(theta):
    return -g*np.sin(theta)/l1


def RungeKutta41(theta, w):
    k1 = f(theta) # w
```

```
    n1 = w # theta
    k2 = f(theta + n1*dt/2)
    n2 = w + k1*dt/2
    k3 = f(theta + n2*dt/2)
    n3 = w + k2*dt/2
    k4 = f(theta + n3*dt)
    n4 = w + k3*dt
    w = w + (k1/6 + k2/3 + k3/3 + k4/6)*dt
    theta = theta + (n1/6 + n2/3 + n3/3 + n4/6)*dt

    w_series.append(w)
    theta_series.append(theta)
    xs.append(l1*np.sin(theta))
    ys.append(-l1*np.cos(theta))
    T = m1*(l1**2)*(w**2)/2
    U = - m1*g*l1*np.cos(theta)
    H = T + U
    T_series.append(T)
    U_series.append(U)
    H_series.append(H)


for i in range(steps):
    theta = theta_series[-1]
    w = w_series[-1]
    RungeKutta41(theta, w)


def guide(t):
    return 3*10**(-9)*t


H_series = np.array(H_series)
H_exact = H_series[0]
ax.plot(t_series, H_exact-H_series, label="$E_0␣-␣E$")
ax.plot(t_series, guide(t_series),
        label="guide:"+R"$f(t)=3*10^{-9}t$", ls="dashed")
ax.set_xlabel("time␣/s", fontsize=14)
ax.grid()
ax.legend()

plt.tight_layout()
```

```
84 fig.savefig('./figure/RK41/evaluation2/{0}-{1}-{2}-{3}{4}{5}.jpeg'
85              .format(dt_now.year, dt_now.month, dt_now.day,
86                      dt_now.hour, dt_now.minute, dt_now.second))
87 # plt.show()
```

4.4　計算機環境

4.5　参考文献