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 11 日

目次

1　はじめに

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．図のキャプションなどで現れる RK4 という文字列は 4 段 4 次の Runge-Kutta 法を表すものとする．
なお今回用いた python のソースコードや掲載することのできなかった図は Google Drive から見られるようにした．

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，

$$k_1 = f(\theta^{(m)}) \tag{4}$$

$$n_1 = \omega^{(m)} \tag{5}$$

$$k_2 = f\Big(\theta^{(m)} + \frac{n_1}{2}\Delta t\Big) \tag{6}$$

$$n_2 = \omega^{(m)} + \frac{k_1}{2}\Delta t \tag{7}$$

$$k_3 = f\Big(\theta^{(m)} + \frac{n_2}{2}\Delta t\Big) \tag{8}$$

$$n_3 = \omega^{(m)} + \frac{k_2}{2}\Delta t \tag{9}$$

$$k_4 = f\Big(\theta^{(m)} + n_3\Delta t\Big) \tag{10}$$

$$n_4 = \omega^{(m)} + k_3\Delta t \tag{11}$$

と定義すると，

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

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

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，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 2，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 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（Google drive）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．
ソースコード 2 では長さ l，質量 m，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，$l = 1, m = 1, \omega_0 = 0$ とした．
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 2 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．

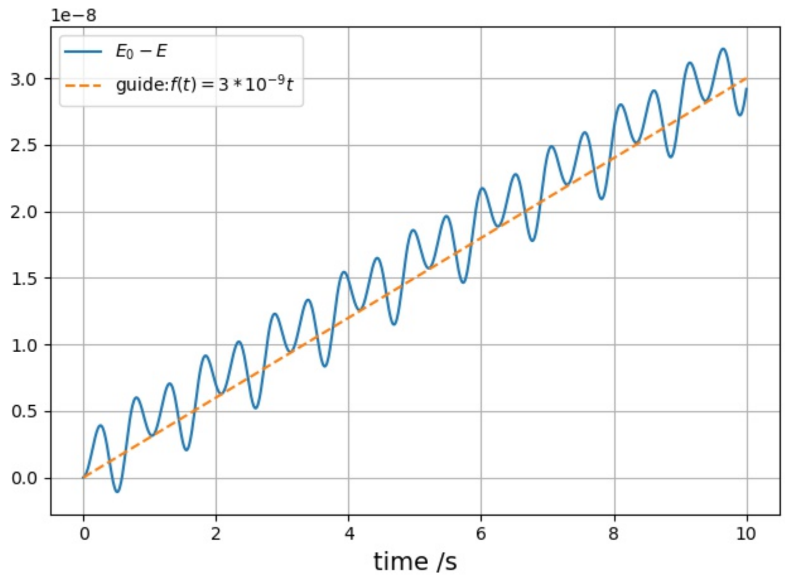

図 1　系のエネルギー誤差の時間変化

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 $\frac{1}{10}$ 倍すれば誤差は $\frac{1}{10000}$ 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 3 において，$l=1$, $m=1$, $\omega_0=0$ としたうえで初角度をそれぞれ $\frac{\pi}{2}$, $\frac{\pi}{3}$, $\frac{\pi}{4}$, $\frac{\pi}{6}$ 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．

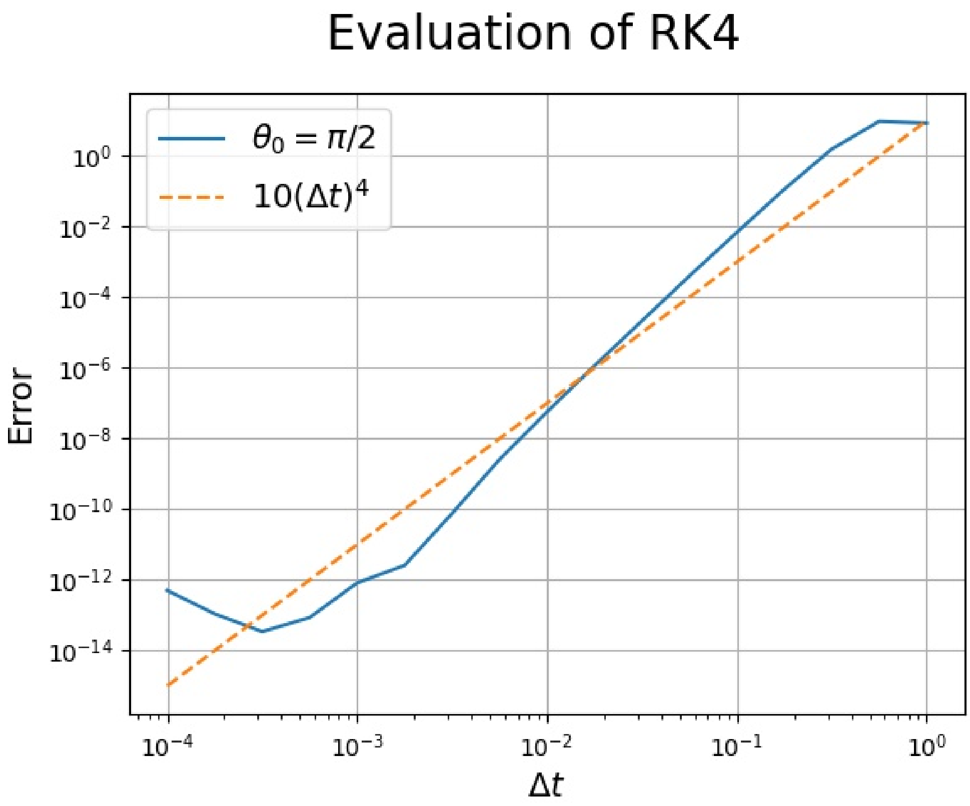

図 2　初角度 $\frac{\pi}{2}$ における力学的エネルギーの精度

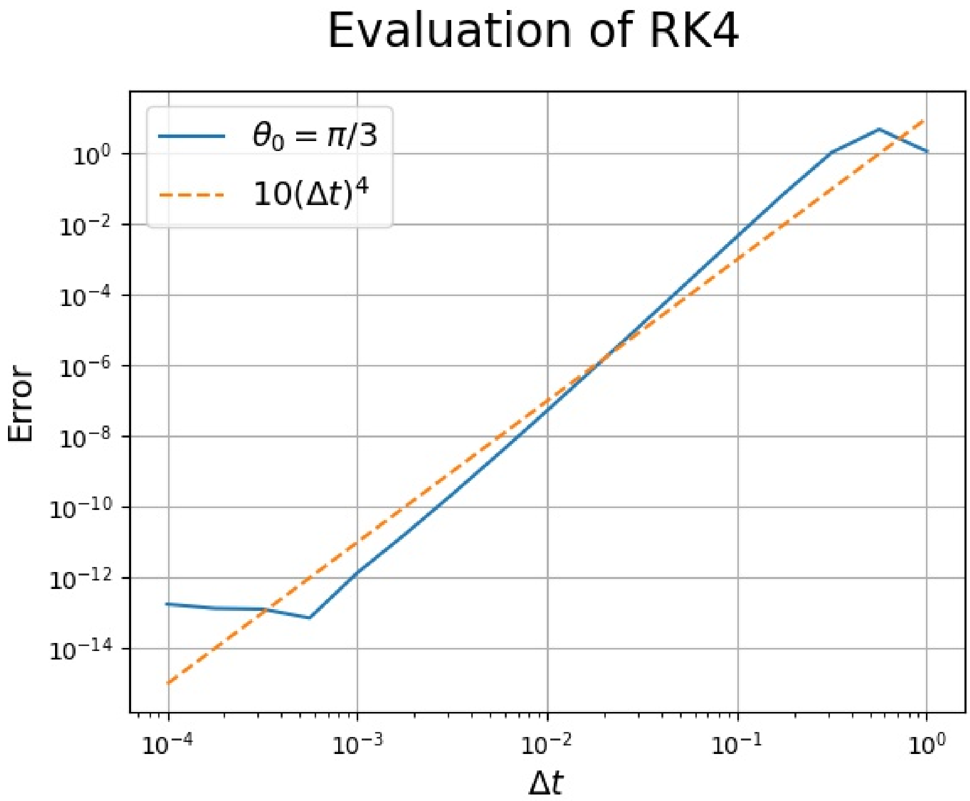

図 3　初角度 $\frac{\pi}{3}$ における力学的エネルギーの精度

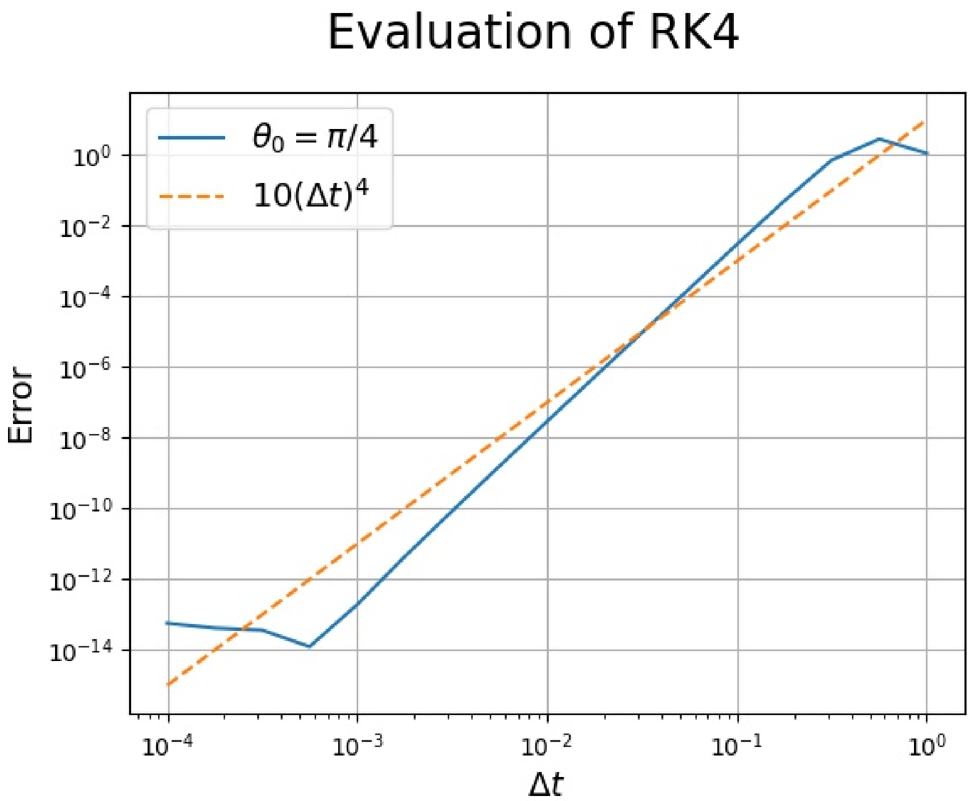

図 4　初角度 $\frac{\pi}{4}$ における力学的エネルギーの精度

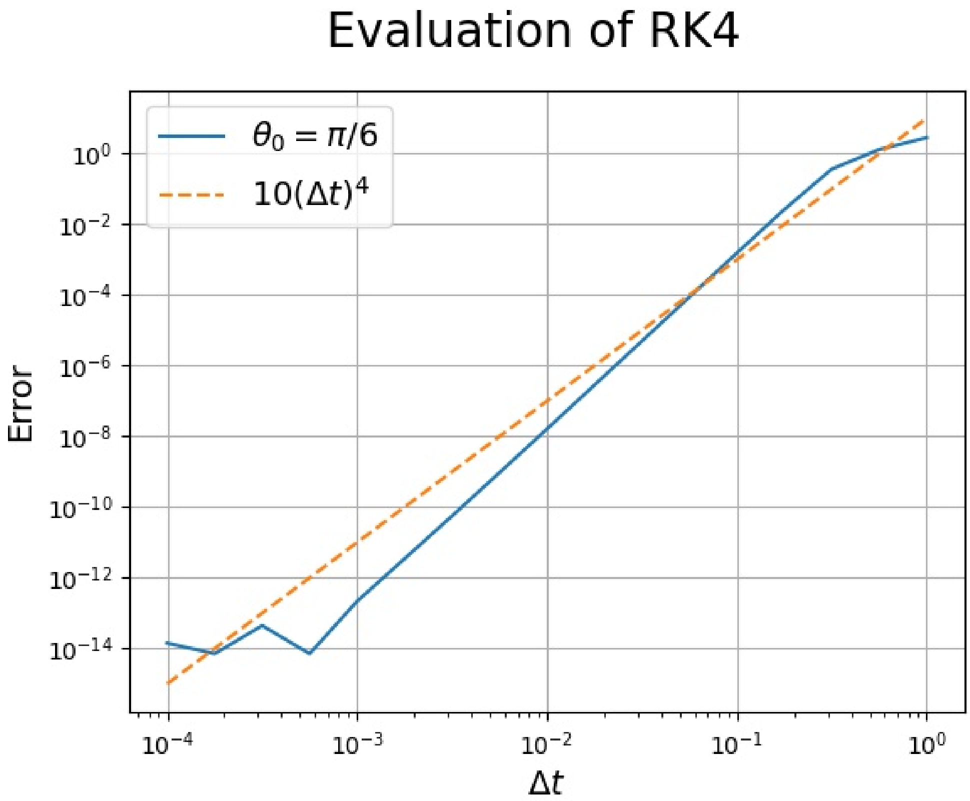

図 5　初角度 $\frac{\pi}{6}$ における力学的エネルギーの精度

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．また，このようなことが成り立つのであれば，誤差の増加が大きいと予想される θ では時間幅 Δt を大きくとるような可変的な時間幅を設定できるプログラムを作成することにより誤差の増加を緩やかにすることができるかもしれない．

3　二重振り子の運動

この章では二重振り子の運動方程式に対して Runge-Kutta 法を適用し，その特性を確認する．全体の流れとしては単振り子のときと同様である．

3.1　二重振り子の運動方程式

ここでは，長さ l_1 で質点の質量が m_1 の単振り子 1 と長さ l_2 で質点の質量が m_2 の単振り子 2 が連結された二重振り子を考える．鉛直下向きと単振り子 1, 2 がなす角をそれぞれ θ_1, θ_2 とする．
このような二重振り子の Lagrangian に対して Euler-Lagrange 方程式を立てると以下のような運動方程式が求まる（導出の過程は 4.1 節に示した）．なお，$\phi = \theta_1 - \theta_2$ とした．

$$
\begin{cases}
\ddot{\theta}_1 = \dfrac{-m_2 l_2 \dot{\theta}_2{}^2 \sin\phi - (m_1+m_2) g \sin\theta_1 - m_2 l_1 \dot{\theta}_1{}^2 \sin\phi\cos\phi + m_2 g \cos\phi \sin\theta_2}{(m_1+m_2) l_1 - m_2 l_1 \cos^2\phi} & \text{(14a)} \\
\ddot{\theta}_2 = \dfrac{m_2 l_2 \dot{\theta}_2{}^2 \cos\phi\sin\phi + (m_1+m_2) g \sin\theta_1 \cos\phi + (m_1+m_2) l_1 \dot{\theta}_1{}^2 \sin\phi - (m_1+m_2) g \sin\theta_2}{(m_1+m_2) l_2 - m_2 l_2 \cos^2\phi} & \text{(14b)}
\end{cases}
$$

これらの式において $\omega_1 = \dot{\theta}_1$, $\omega_2 = \dot{\theta}_2$ とすると以下のような連立一階微分方程式となる．

$$
\begin{cases}
\dot{\theta}_1 = \omega_1 & \text{(15a)} \\
\ddot{\theta}_1 = \dfrac{-m_2 l_2 \omega_2^2 \sin\phi - (m_1+m_2) g \sin\theta_1 - m_2 l_1 \omega_1^2 \sin\phi\cos\phi + m_2 g \cos\phi \sin\theta_2}{(m_1+m_2) l_1 - m_2 l_1 \cos^2\phi} & \text{(15b)} \\
\dot{\theta}_2 = \omega_2 & \text{(15c)} \\
\ddot{\theta}_2 = \dfrac{m_2 l_2 \omega_2^2 \cos\phi\sin\phi + (m_1+m_2) g \sin\theta_1 \cos\phi + (m_1+m_2) l_1 \omega_1^2 \sin\phi - (m_1+m_2) g \sin\theta_2}{(m_1+m_2) l_2 - m_2 l_2 \cos^2\phi} & \text{(15d)}
\end{cases}
$$

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

以下では式 (15) に対して Runge-Kutta 法を適用することを考える．

$$
f(\theta_1, \theta_2, \omega_1, \omega_2) = \frac{-m_2 l_2 \omega_2^2 \sin\phi - (m_1+m_2) g \sin\theta_1 - m_2 l_1 \omega_1^2 \sin\phi\cos\phi + m_2 g \cos\phi \sin\theta_2}{(m_1+m_2) l_1 - m_2 l_1 \cos^2\phi} \tag{16}
$$

$$
h(\theta_1, \theta_2, \omega_1, \omega_2) = \frac{m_2 l_2 \omega_2^2 \cos\phi\sin\phi + (m_1+m_2) g \sin\theta_1 \cos\phi + (m_1+m_2) l_1 \omega_1^2 \sin\phi - (m_1+m_2) g \sin\theta_2}{(m_1+m_2) l_2 - m_2 l_2 \cos^2\phi} \tag{17}
$$

として，

$$k_{11} = f(\theta_1^{(m)}, \theta_2^{(m)}, \omega_1^{(m)}, \omega_2^{(m)}) \tag{18}$$

$$n_{11} = \omega_1^{(m)} \tag{19}$$

$$k_{21} = h(\theta_1^{(m)}, \theta_2^{(m)}, \omega_1^{(m)}, \omega_2^{(m)}) \tag{20}$$

$$n_{21} = \omega_2^{(m)} \tag{21}$$

$$k_{12} = f\left(\theta_1^{(m)} + n_{11}\frac{\Delta t}{2},\ \theta_2^{(m)} + n_{21}\frac{\Delta t}{2},\ \omega_1^{(m)} + k_{11}\frac{\Delta t}{2},\ \omega_2^{(m)} + k_{21}\frac{\Delta t}{2}\right) \tag{22}$$

$$n_{12} = \omega_1^{(m)} + k_{11}\frac{\Delta t}{2} \tag{23}$$

$$k_{22} = h\left(\theta_1^{(m)} + n_{11}\frac{\Delta t}{2},\ \theta_2^{(m)} + n_{21}\frac{\Delta t}{2},\ \omega_1^{(m)} + k_{11}\frac{\Delta t}{2},\ \omega_2^{(m)} + k_{21}\frac{\Delta t}{2}\right) \tag{24}$$

$$n_{22} = \omega_2^{(m)} + k_{21}\frac{\Delta t}{2} \tag{25}$$

$$k_{13} = f\left(\theta_1^{(m)} + n_{12}\frac{\Delta t}{2},\ \theta_2^{(m)} + n_{22}\frac{\Delta t}{2},\ \omega_1^{(m)} + k_{12}\frac{\Delta t}{2},\ \omega_2^{(m)} + k_{22}\frac{\Delta t}{2}\right) \tag{26}$$

$$n_{13} = \omega_1^{(m)} + k_{12}\frac{\Delta t}{2} \tag{27}$$

$$k_{23} = h\left(\theta_1^{(m)} + n_{12}\frac{\Delta t}{2},\ \theta_2^{(m)} + n_{22}\frac{\Delta t}{2},\ \omega_1^{(m)} + k_{12}\frac{\Delta t}{2},\ \omega_2^{(m)} + k_{22}\frac{\Delta t}{2}\right) \tag{28}$$

$$n_{23} = \omega_2^{(m)} + k_{22}\frac{\Delta t}{2} \tag{29}$$

$$k_{14} = f\left(\theta_1^{(m)} + n_{13}\Delta t,\ \theta_2^{(m)} + n_{23}\Delta t,\ \omega_1^{(m)} + k_{13}\Delta t,\ \omega_2^{(m)} + k_{23}\Delta t\right) \tag{30}$$

$$n_{14} = \omega_1^{(m)} + k_{13}\Delta t \tag{31}$$

$$k_{24} = f\left(\theta_1^{(m)} + n_{13}\Delta t,\ \theta_2^{(m)} + n_{23}\Delta t,\ \omega_1^{(m)} + k_{13}\Delta t,\ \omega_2^{(m)} + k_{23}\Delta t\right) \tag{32}$$

$$n_{24} = \omega_2^{(m)} + k_{23}\Delta t \tag{33}$$

と定義すると，

$$\omega_1^{(m+1)} = \omega_1^{(m)} + \left(\frac{1}{6}k_{11} + \frac{1}{3}k_{12} + \frac{1}{3}k_{13} + \frac{1}{6}k_{14}\right)\Delta t \tag{34}$$

$$\theta_1^{(m+1)} = \theta_1^{(m)} + \left(\frac{1}{6}n_{11} + \frac{1}{3}n_{12} + \frac{1}{3}n_{13} + \frac{1}{6}n_{14}\right)\Delta t \tag{35}$$

$$\omega_2^{(m+1)} = \omega_2^{(m)} + \left(\frac{1}{6}k_{11} + \frac{1}{3}k_{12} + \frac{1}{3}k_{13} + \frac{1}{6}k_{14}\right)\Delta t \tag{36}$$

$$\theta_2^{(m+1)} = \theta_2^{(m)} + \left(\frac{1}{6}n_{21} + \frac{1}{3}n_{22} + \frac{1}{3}n_{23} + \frac{1}{6}n_{24}\right)\Delta t \tag{37}$$

と表せる．

単振り子と同様に，今回はこれを Python によって実装し，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 5，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 6，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 7 に示す．

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，3.2.1 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，簡単のために $l = l_1 = l_2$ とし，$\theta_1, \theta_2 << 1$ 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．このとき解析解は，

$$\theta_1 = \frac{2-\sqrt{2}}{4000}\cos\left(\sqrt{(2+\sqrt{2})\frac{g}{l}}t\right) + \frac{2+\sqrt{2}}{4000}\cos\left(\sqrt{(2-\sqrt{2})\frac{g}{l}}t\right) \tag{38}$$

$$\theta_2 = -\frac{2\sqrt{2}-2}{4000}\cos\left(\sqrt{(2+\sqrt{2})\frac{g}{l}}t\right) + \frac{2\sqrt{2}+2}{4000}\cos\left(\sqrt{(2-\sqrt{2})\frac{g}{l}}t\right) \tag{39}$$

と書くことができる．

Runge-Kutta 法によって求まる数値解とこの解析解を比較するためのプログラムはソースコード 4 に示した．このコードを実行すると以下のようなグラフが得られる（ここでは特に $l = 1$ とした）．

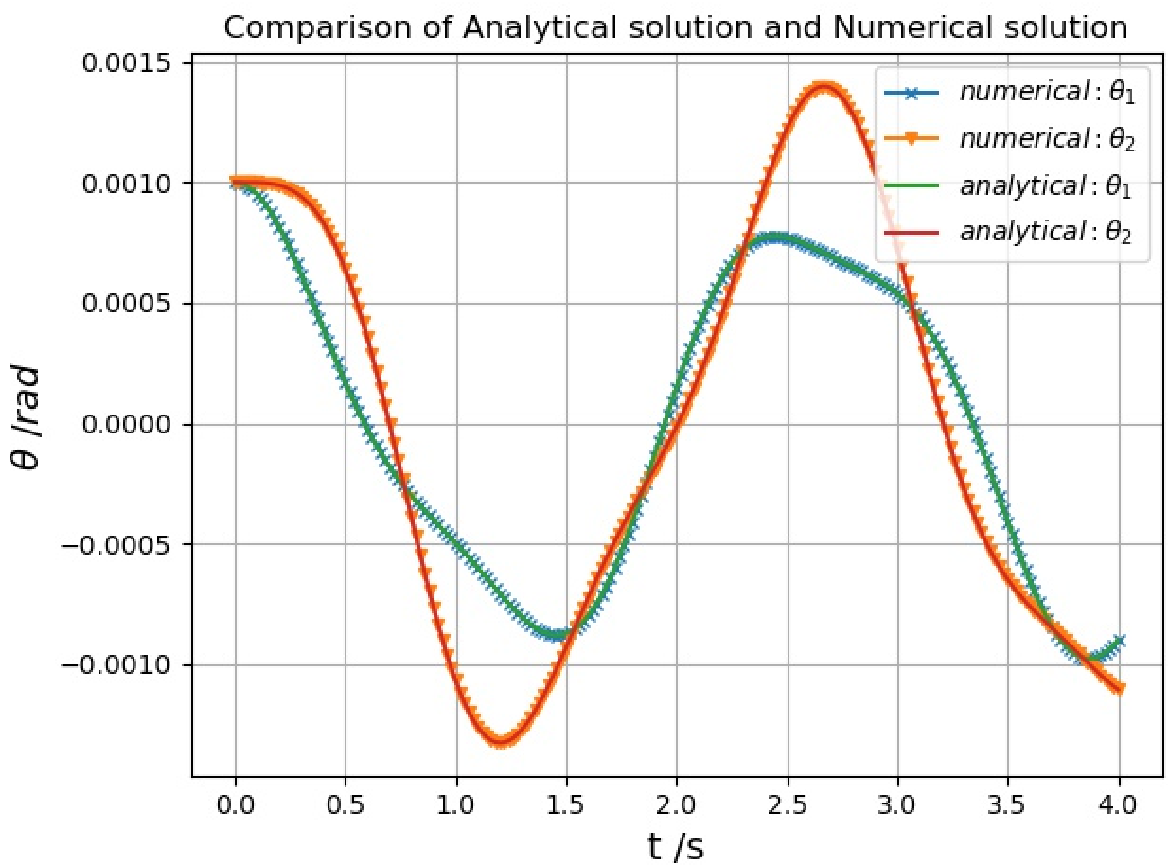

図 6　二重振り子の微小角近似における解析解と RK4 による数値解の比較

この図より，微小角近似における解析解と Runge-Kutta 法による数値解は十分に一致しているため 3.2.1 節で定めた式が妥当であると考えることができる．

3.3 結果

3.3.1 二重振り子の運動のアニメーション

ソースコード 5 では長さ l_1, l_2，質量 m_1, m_2，初角度 θ_{10}, θ_{20}，初角速度 ω_{10}, ω_{20} 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，$l_1 = l_2 = 1, m_1 = m_2 = 1, \omega_{10} = \omega 20 = 0$ として運動の様子を見る．また，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（Google drive）に保存した．

- 初角度 $\theta_{10} = \dfrac{\pi}{2}$, $\theta_{20} = \dfrac{5\pi}{4}$ の二重振り子
- 初角度 $\theta_{10} = \dfrac{\pi}{4}$, $\theta_{20} = \dfrac{\pi}{3}$ の二重振り子
- 初角度 $\theta_{10} = \dfrac{\pi}{2}$, $\theta_{20} = \dfrac{\pi}{2}$ の二重振り子

3.3.2 エネルギー誤差の時間変化

この二重振り子の系では力学的エネルギーが保存されるが，Runge-Kutta 法を用いて数値計算をした場合には単振り子の場合と同じように，系全体のエネルギーが保存されるとは限らない．そのため，ここではいくつかの初期条件に対して時間経過とともに系全体のエネルギーがどのように変化するのかをグラフにまとめる．

以下に示す図において，横軸はすべて時間を表し，左上のグラフは系全体のエネルギーの変化，右上のグラフは振り子 1, 2 の運動エネルギー (T_1，T_2) とポテンシャルエネルギー (U_1，U_2) の変化，左下のグラフは振り子 1, 2 の力学的エネルギー (H_1, H_2) の変化，右下のグラフは，真の力学的エネルギーを E_0，計算された力学的エネルギーを E としたときの，$\log\left|1-\dfrac{E}{E_0}\right|$ の変化をそれぞれ表している．なお，この図を作成するプログラムはソースコード 6 であり，パラメータとして l_1，l_2，m_1, m_2，θ_{10}，θ_{20}，ω_{10}，ω_{20} を設定することができるが，以下に示す図では，$l_1 = l_2 = 1.0$, $m_1 = m_2 = 1.0$, $\omega_{10} = \omega_{20} = 0$ と固定している．

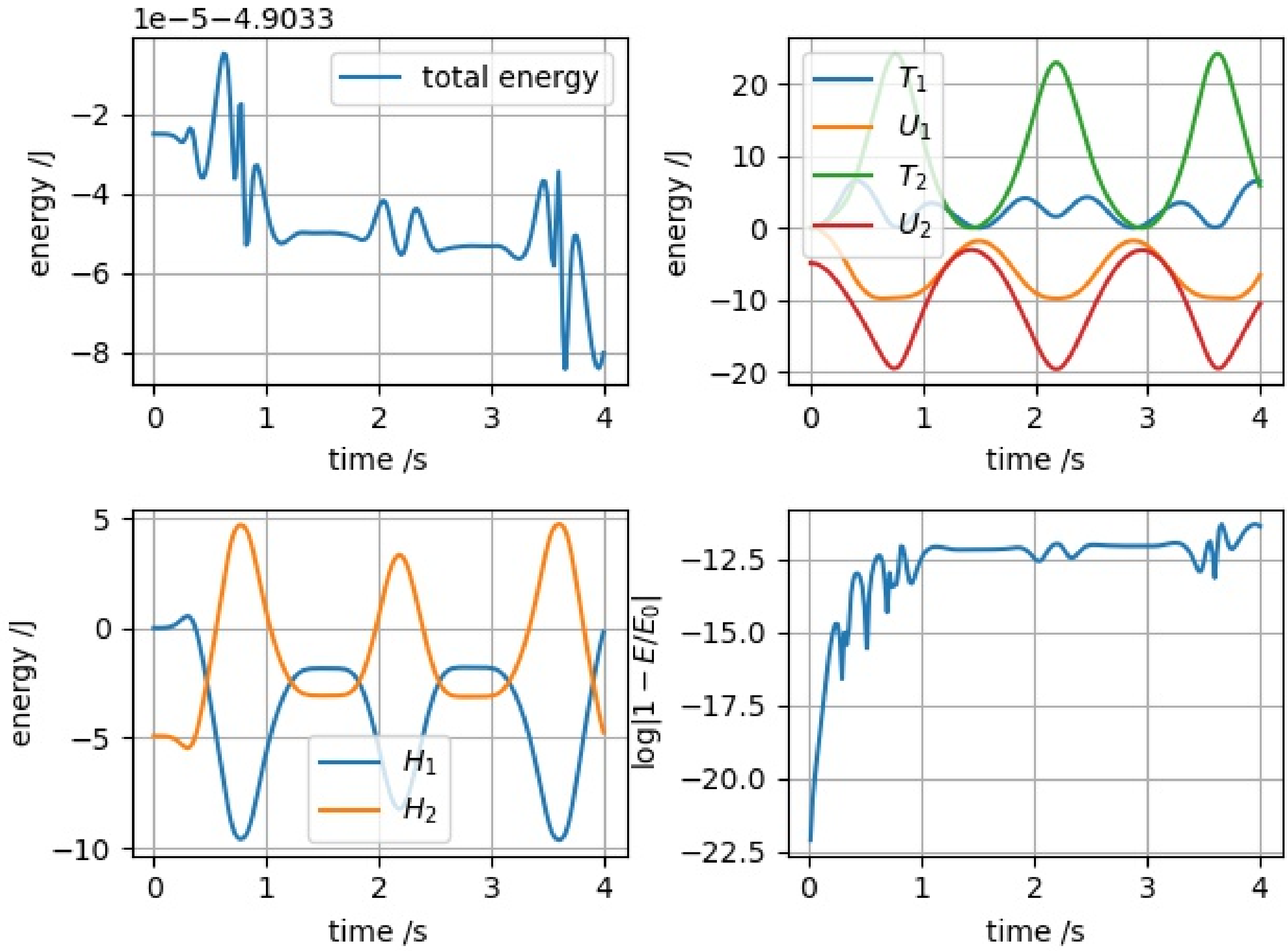

図 7　$\theta_{10} = \dfrac{\pi}{2}$，$\theta_{20} = \dfrac{\pi}{3}$ のときのエネルギーの時間変化

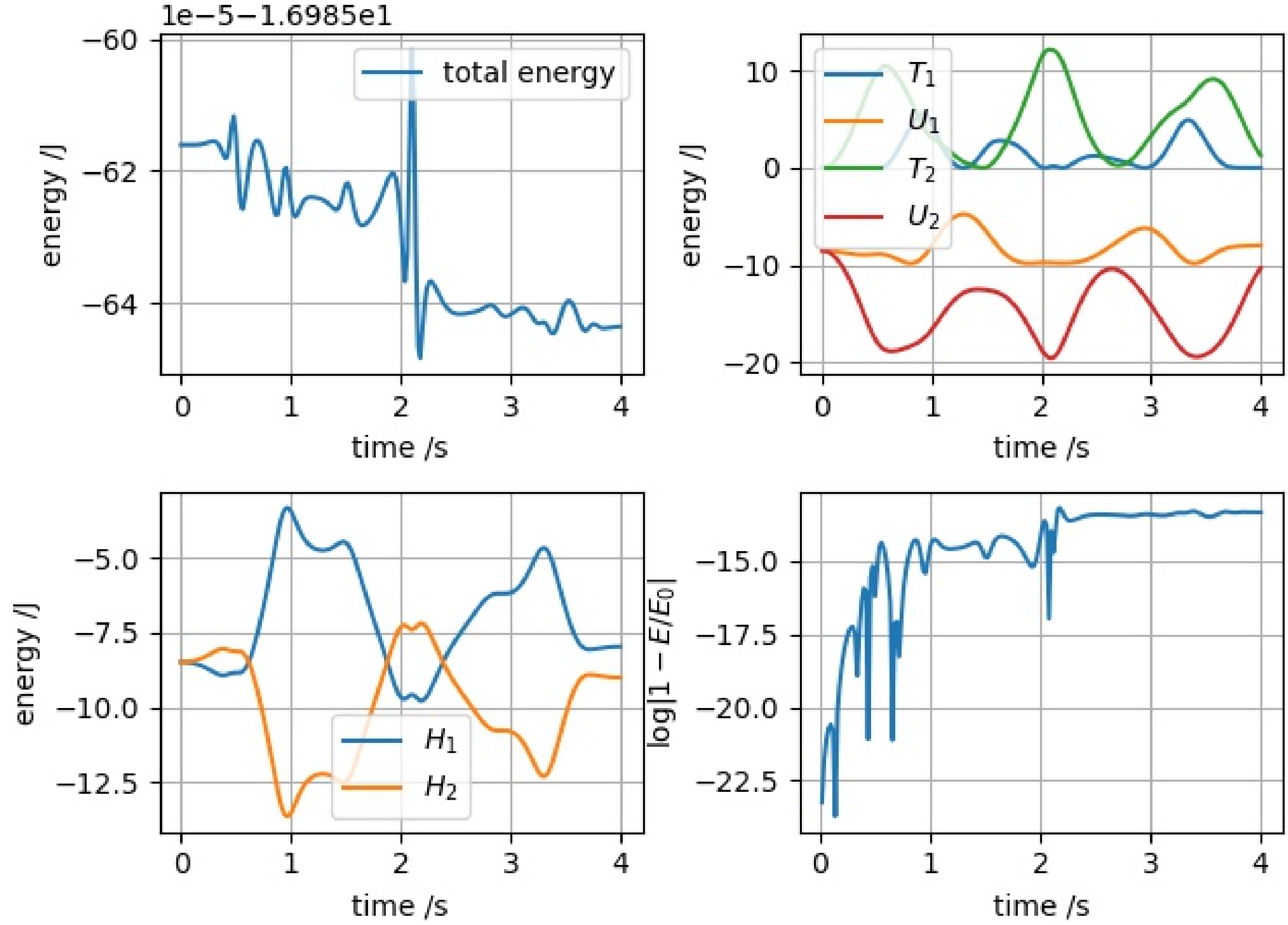

図 8　$\theta_{10} = \dfrac{\pi}{6}$, $\theta_{20} = \dfrac{\pi}{3}$ のときのエネルギーの時間変化

これらの図を見ると，系全体のエネルギーは計算の誤差により 10^{-5} のオーダーで減少していっていることがわかる.

3.3.3　時間幅の取り方による誤差の変化

ここでは単振り子の場合と同じように Runge-Kutta 法における時間幅 Δt の取り方によってエネルギーの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 9 から図 10 はソースコード 7 において，$l_1 = l_2 = 1.0$, $m_1 = m_2 = 1.0$, $\omega_{10} = \omega_{20} = 0$ としたうえで初角度をそれぞれ $\theta_{10} = \frac{\pi}{2}$, $\theta_{20} = \frac{\pi}{3}$, $\theta_{10} = \frac{\pi}{6}$, $\theta_{20} = \frac{\pi}{3}$ と設定して得られたグラフである．なお，ここでの誤差は 4 秒経過後の $|E - E_0|$ と定義した．

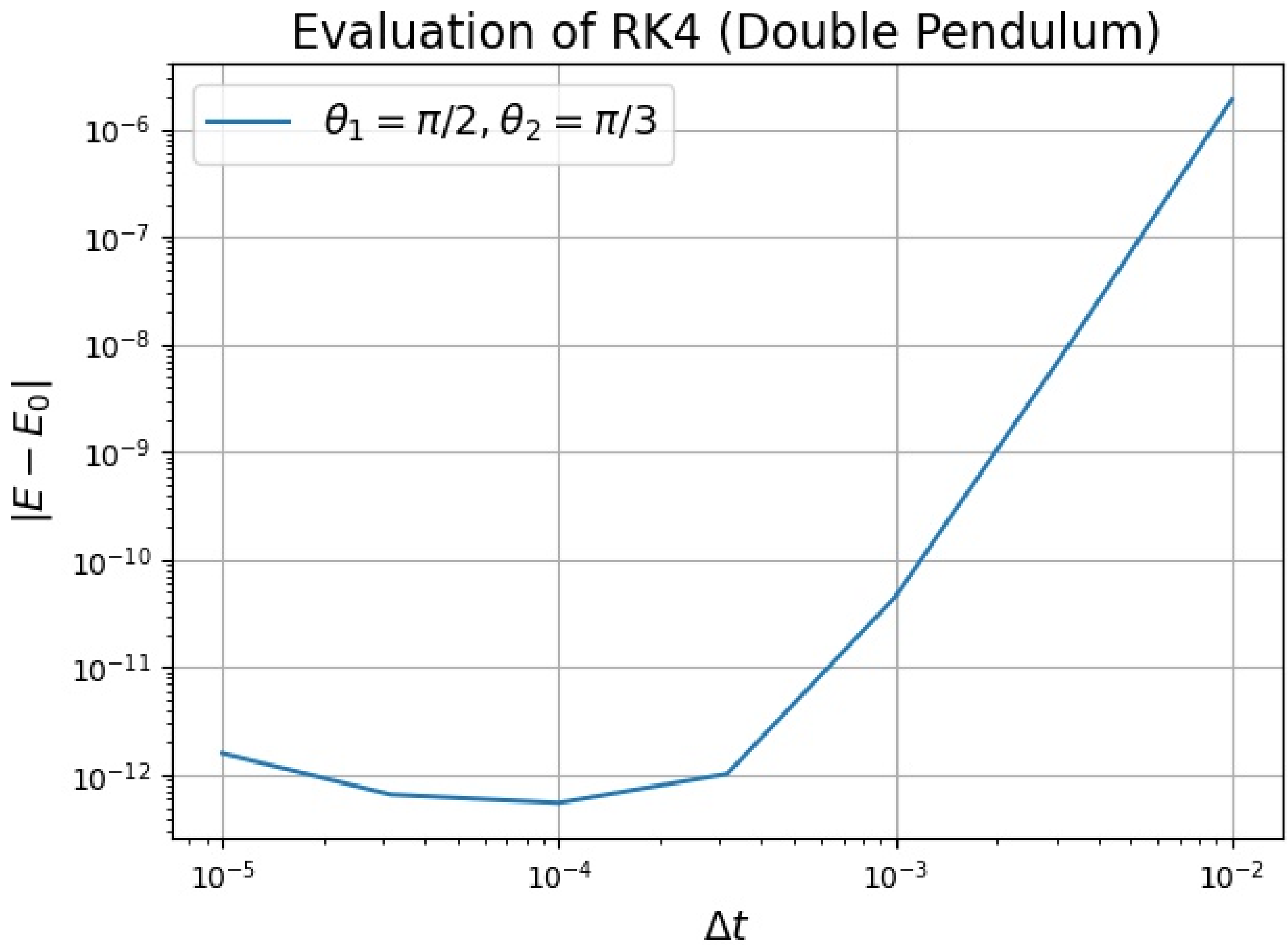

図 9　$\theta_{10} = \frac{\pi}{2}$, $\theta_{20} = \frac{\pi}{3}$ としたときの RK4 の次数

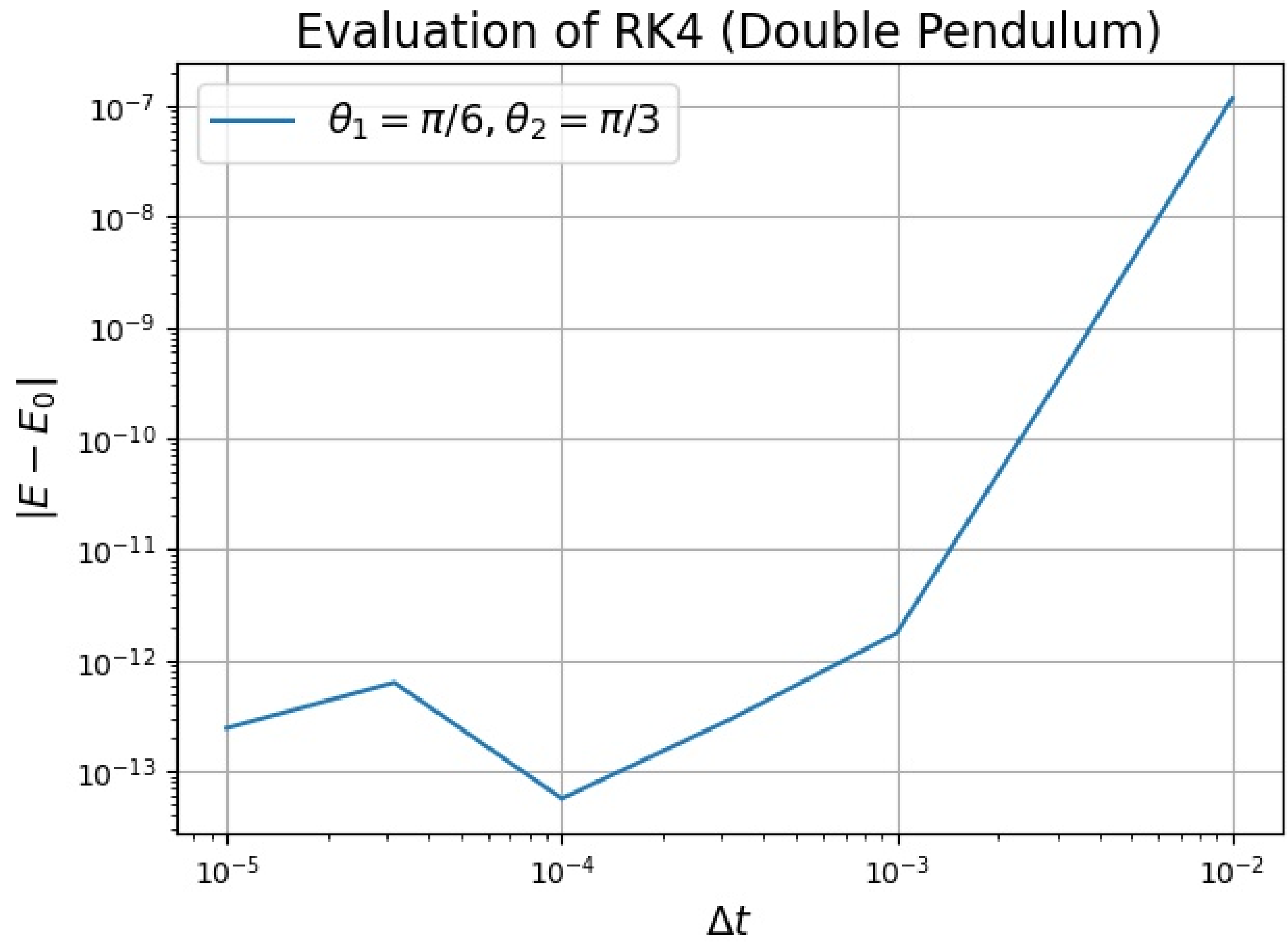

図 10　$\theta_{10} = \dfrac{\pi}{6}$，$\theta_{20} = \dfrac{\pi}{3}$ としたときの RK4 の次数

これらの図を見て言えることとしては，$10^{-3} \leq \Delta t \leq 10^{-2}$ の範囲では期待される 4 次の精度が出ているが，それよりも Δt を小さくとると 4 次の精度を達成しているとはいえないということである．これは，ここに示した初角度の設定に限定されるものではなく，ほかの初角度の組み合わせにおいても全体的に同じ傾向が見られた．

3.4 考察

まずアニメーションについてであるが，二重振り子はカオス運動を起こすということから人間が簡単に予想できるほど単純な動きはせずこのアニメーションが正しく現実の二重振り子を再現できているかは判断することができない．強いて言うとするならば，質点の位置が連続的に変化していることは確かめられるため，Runge-Kutta 法の計算が正しく実行されていると推測できる．

次に，系全体のエネルギーが時間経過とともにどのように変化するかについてであるが，これについては単振り子の場合と同じように，時間経過とともに計算される系全体のエネルギーは小さくなっていき正しい値からは離れていくという結果が得られた．大域的に見たときはエネルギーが減少しているということは単振り子の場合と共通しているが，一方で局所的に見たときのエネルギー変化については相違点がある．単振り子ではエネルギーの変化が単純な振動をしているように捉えることができたが，二重振り子では少し複雑な振動を繰り返しているように見える．
これは単振り子では角度 θ と角速度 ω が単純なトレードオフ関係にあったことに対し，二重振り子ではそうではないということによるのではないかと考えた．これはこの後述べる誤差の Δt 依存性の議論にも共通する点である．

最後に，誤差の Δt 依存性についてである．Δt を小さくとりすぎると誤差が大きくなってしまうという現象は単振り子の場合にも共通しており，これらの運動に対して Runge-Kutta 法を適用したときの性質には類似性があると考えられる．単振り子でこのような傾向があることの理由として 2.4 節では，誤差の広がりやすさが θ に依存しており，Δt を小さくすると誤差が広がりやすい θ の近傍での計算回数が増えるためではないかということを述べた．
二重振り子の場合についても単振り子の場合と同じように誤差の広がり方がある程度 θ に依存するのであれば，このような現象は説明できると考えられる．しかし，二重振り子は単振り子ほど運動が単純ではないため，単振り子と二重振り子を簡単に比較することは望ましくない．特に，単振り子では θ が大きいならば ω が小さいということが力学的エネルギー保存則から簡単に言えたが，二重振り子については片方の振り子に注目したとき，θ が大きく，かつ ω も大きいという状況がありうるので単振り子ほど単純な説明ができるとは限らない．
これを確かめるためには誤差の広がり方の θ および ω に対する依存性を確かめるプログラムを作成すればよいと考えられるが，今回は時間不足のため確認できなかった．

今回新たに生まれた疑問に関しては明確な理由を述べることができなかったが，はじめに述べた目的である「系全体のエネルギーが Runge-Kutta 法によってどれくらい保存されるのか」を確かめるということについては $\Delta t = 10^{-3}$ 程度までは 4 次の精度をもち，$\Delta t = 10^{-3}$ 程度とすれば単振り子・二重振り子いずれの場合についてもエネルギーの誤差は 10^{-10} 以下まで抑えることができると分かった．

4　付録

4.1　二重振り子の運動方程式の導出

長さ l_1，質量 m_1 の振り子の下に長さ l_2, 質量 m_2 の振り子を連結した二重振り子を考える．
振り子 1, 2 が鉛直下向きとなす角を θ_1，θ_2 としてデカルト座標で表すと，

$$(x_1,\ y_1) = (l_1 \sin\theta_1,\ -l_1\cos\theta_1) \tag{40}$$

$$(x_2,\ y_2) = (l_1 \sin\theta_1 + l_2\sin\theta_2,\ -l_1\cos\theta_1 - l_2\cos\theta_2) \tag{41}$$

となり，これを時間微分すると，

$$(\dot{x_1},\ \dot{y_1}) = (l_1\dot{\theta_1}\cos\theta_1,\ l_1\dot{\theta_1}\sin\theta_1) \tag{42}$$

$$(\dot{x_2},\ \dot{y_2}) = (l_1\dot{\theta_1}\cos\theta_1 + l_2\dot{\theta_2}\cos\theta_2,\ l_1\dot{\theta_1}\sin\theta_1 + l_2\dot{\theta_2}\sin\theta_2) \tag{43}$$

となる．
ふりこ i の運動エネルギー，ポテンシャルエネルギーをそれぞれ T_i，U_i と表すとこの系の Lagrangian は，

$$\begin{aligned} L &= T_1 + T_2 - U_1 - U_2 \\ &= \frac{1}{2}m_1(\dot{x_1}^2 + \dot{y_1}^2) + \frac{1}{2}m_2(\dot{x_2}^2 + \dot{y_2}^2) - m_1 g y_1 - m_2 g y_2 \\ &= \frac{1}{2}m_1 l_1^2\dot{\theta_1}^2 + \frac{1}{2}m_2\left[l_1^2\dot{\theta_1}^2 + l_2^2\dot{\theta_2}^2 + 2l_1l_2\dot{\theta_1}\dot{\theta_2}\cos\phi\right] + m_1gl_1\cos\theta_1 + m_2g(l_1\cos\theta_1 + l_2\cos\theta_2) \end{aligned} \tag{44}$$

となる．ただし $\theta_1 - \theta_2 = \phi$ とした．これを Euler-Lagrange 方程式に代入すると，

$$\begin{pmatrix} (m_1+m_2)l_1 & m_2l_2\cos\phi \\ l_1\cos\phi & l_2 \end{pmatrix}\begin{pmatrix} \ddot{\theta_1} \\ \ddot{\theta_2} \end{pmatrix} = \begin{pmatrix} -m_2l_2\dot{\theta_2}^2\sin\phi - (m_1+m_2)g\sin\theta_1 \\ l_1\dot{\theta_1}^2\sin\phi - g\sin\theta_2 \end{pmatrix} \tag{45}$$

が求まる．ここで，

$$\begin{pmatrix} (m_1+m_2)l_1 & m_2l_2\cos\phi \\ l_1\cos\phi & l_2 \end{pmatrix}^{-1} = \frac{1}{(m_1+m_2)l_1l_2 - m_2l_1l_2\cos^2\phi}\begin{pmatrix} l_2 & -m_2l_2\cos\phi \\ -l_1\cos\phi & (m_1+m_2)l_1 \end{pmatrix} \tag{46}$$

であることから，

$$\begin{cases} \ddot{\theta_1} = \dfrac{-m_2l_2\dot{\theta_2}^2\sin\phi - (m_1+m_2)g\sin\theta_1 - m_2l_1\dot{\theta_1}^2\sin\phi\cos\phi + m_2g\cos\phi\sin\theta_2}{(m_1+m_2)l_1 - m_2l_1\cos^2\phi} & \text{(47a)} \\[2ex] \ddot{\theta_2} = \dfrac{m_2l_2\dot{\theta_2}^2\cos\phi\sin\phi + (m_1+m_2)g\sin\theta_1\cos\phi + (m_1+m_2)l_1\dot{\theta_1}^2\sin\phi - (m_1+m_2)g\sin\theta_2}{(m_1+m_2)l_2 - m_2l_2\cos^2\phi} & \text{(47b)} \end{cases}$$

が求まる．

4.2 計算機環境

- CPU: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz 3.60 GHz
- メモリ: 16.0 GB
- OS: Windows10 Home
- プログラミング言語: Python 3.8.5

4.3 ソースコード

今回用いたプログラムは Google Drive で見られるようにしたが，ここにも掲載する．

ソースコード 1 単振り子のアニメーション作成

```python
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]]
    image = ax.plot(x, y, ’o-’, lw=2, c="black", label="pendulum")
    ax.grid(True)
    ax.axis(’equal’)
    ax.set_title("simple␣pendulum", fontsize=18)
    images.append(image)

ani = anim.ArtistAnimation(fig, images, interval=10)
plt.tight_layout()

ani.save(’./figure/RK41/animation/{0}-{1}-{2}-{3}{4}{5}.gif’.format(
    dt_now.year, dt_now.month, dt_now.day,
    dt_now.hour, dt_now.minute, dt_now.second), writer=’pillow’, fps=50)
# plt.show()
```

ソースコード 2　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()
fig.savefig('./figure/RK41/evaluation2/{0}-{1}-{2}-{3}{4}{5}.jpeg'
            .format(dt_now.year, dt_now.month, dt_now.day,
                    dt_now.hour, dt_now.minute, dt_now.second))
# plt.show()
```

ソースコード 3　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)

```

```python

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

ソースコード 4　二重振り子における微小角近似の解析解と Rk4 の比較

```
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()
l1 = 1.0
l2 = 1.0
m1 = 1.0
m2 = 1.0
theta10 = 0.001
theta20 = 0.001
w10 = 0.0
w20 = 0.0
C_plus = (2-np.sqrt(2))/4000
C_minus = (2+np.sqrt(2))/4000
w_plus = np.sqrt((2+np.sqrt(2))*g/l1)
w_minus = np.sqrt((2-np.sqrt(2))*g/l1)


# parameters
t_start = 0
t_end = 4
steps = 200


# calculated
dt = (t_end - t_start)/steps


# lists
t_series = np.linspace(t_start, t_end, steps+1)
theta1_series = deque([theta10])
theta2_series = deque([theta20])
w1_series = deque([w10])
w2_series = deque([w20])
T1_series = deque([m1*(l1**2)*(w10**2)/2])
T2_series = deque([(m2*((l1**2)*(w10**2)+(l2**2)*(w20**2)
                  + 2*l1*l2*np.cos(theta10-theta20)*w10*w20))/2])
U1_series = deque([-m1*g*l1*np.cos(theta10)])
U2_series = deque([-m2*g*(l1*np.cos(theta10)+l2*np.cos(theta20))])
H_series = deque([T1_series[0]+T2_series[0]+U1_series[0]+U2_series[0]])


def f(theta1, theta2, w1, w2):
```

```
    phi = (theta1 - theta2) % (2*np.pi)
    numerator = (-m2*l2*(w2**2)*np.sin(phi)
                 - (m1+m2)*g*np.sin(theta1)
                 - m2*l1*(w1**2)*np.sin(phi)*np.cos(phi)
                 + m2*g*np.cos(phi)*np.sin(theta2))
    denominator = (m1+m2)*l1-m2*l1*((np.cos(phi))**2)
    value = numerator/denominator
    return value


def h(theta1, theta2, w1, w2):
    phi = (theta1-theta2) % (2*np.pi)
    numerator = (m2*l2*(w2**2)*np.cos(phi)*np.sin(phi)
                 + (m1+m2)*g*np.sin(theta1)*np.cos(phi)
                 + (m1+m2)*l1*(w1**2)*np.sin(phi)
                 - (m1+m2)*g*np.sin(theta2))
    denominator = (m1+m2)*l2 - m2*l2*((np.cos(phi)**2))
    value = numerator/denominator
    return value


def exact_theta1(t):
    value = C_plus*np.cos(w_plus*t)+C_minus*np.cos(w_minus*t)
    return value


def exact_theta2(t):
    value = (- np.sqrt(2)*C_plus*np.cos(w_plus*t)
             + np.sqrt(2)*C_minus*np.cos(w_minus*t))
    return value


def RungeKutta42(theta1, theta2, w1, w2, dt):
    k11 = f(theta1, theta2, w1, w2) # w1
    n11 = w1 # theta1
    k21 = h(theta1, theta2, w1, w2) # w2
    n21 = w2 # theta2

    k12 = f(theta1+n11*dt/2, theta2+n21*dt/2, w1+k11*dt/2, w2+k21*dt/2)
    n12 = w1 + k11*dt/2
    k22 = h(theta1+n11*dt/2, theta2+n21*dt/2, w1+k11*dt/2, w2+k21*dt/2)
    n22 = w2 + k21*dt/2

    k13 = f(theta1+n12*dt/2, theta2+n22*dt/2, w1+k12*dt/2, w2+k22*dt/2)
    n13 = w1 + k12*dt/2
    k23 = h(theta1+n12*dt/2, theta2+n22*dt/2, w1+k12*dt/2, w2+k22*dt/2)
    n23 = w2 + k22*dt/2

    k14 = f(theta1+n13*dt, theta2+n23*dt, w1+k13*dt, w2+k23*dt)
```

```
    n14 = w1 + k13*dt
    k24 = h(theta1+n13*dt, theta2+n23*dt, w1+k13*dt, w2+k23*dt)
    n24 = w2 + k23*dt

    w1 = w1 + (k11/6 + k12/3 + k13/3 + k14/6)*dt
    theta1 = theta1 + (n11/6 + n12/3 + n13/3 + n14/6)*dt

    w2 = w2 + (k21/6 + k22/3 + k23/3 + k24/6)*dt
    theta2 = theta2 + (n21/6 + n22/3 + n23/3 + n24/6)*dt

    w1_series.append(w1)
    w2_series.append(w2)

    theta1_series.append(theta1)
    theta2_series.append(theta2)

    phi = (theta1-theta2) % (2*np.pi)

    T1 = m1*(l1**2)*(w1**2)/2
    T2 = (m2*((l1**2)*(w1**2)+(l2**2)*(w2**2)
              + 2*l1*l2*np.cos(phi)*w1*w2))/2
    T1_series.append(T1)
    T2_series.append(T2)

    U1 = -m1*g*l1*np.cos(theta1)
    U2 = -m2*g*(l1*np.cos(theta1)+l2*np.cos(theta2))
    U1_series.append(U1)
    U2_series.append(U2)

    H = T1 + T2 + U1 + U2
    H_series.append(H)


for i in range(steps):
    theta1 = theta1_series[-1]
    theta2 = theta2_series[-1]
    w1 = w1_series[-1]
    w2 = w2_series[-1]
    RungeKutta42(theta1, theta2, w1, w2, dt)


# plot
fig, ax = plt.subplots()

ax.set_xlabel("t /s", fontsize=14)
ax.set_ylabel(R"$\theta\ /rad$", fontsize=14)
ax.plot(t_series, theta1_series, label=R"$numerical:\theta_1$", marker='x', ms=4)
ax.plot(t_series, theta2_series, label=R"$numerical:\theta_2$", marker='v', ms=4)
ax.plot(t_series, exact_theta1(t_series), label=R"$analytical:\theta_1$")
```

```
ax.plot(t_series, exact_theta2(t_series), label=R"$analytical:\theta_2$")
ax.grid()
ax.set_title("Comparison␣of␣Analytical␣solution␣and␣Numerical␣solution")
plt.legend()
plt.tight_layout()

fig.savefig('./figure/RK42/comparison1/{0}-{1}-{2}-{3}{4}{5}.jpeg'
            .format(dt_now.year, dt_now.month, dt_now.day, dt_now.hour,
                    dt_now.minute, dt_now.second))
# plt.show()
```

ソースコード 5　二重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

# parameters
l1 = 1.0
l2 = 1.0
m1 = 1.0
m2 = 1.0
theta10 = np.pi/4
theta20 = np.pi/3
w10 = 0.0
w20 = 0.0
t_start = 0
t_end = 20
steps = 2000

# calculated
dt = (t_end - t_start)/steps

# lists
theta1_series = deque([theta10])
theta2_series = deque([theta20])
w1_series = deque([w10])
w2_series = deque([w20])
x1s = deque([l1*np.sin(theta10)])
y1s = deque([-l1*np.cos(theta10)])
x2s = deque([l1*np.sin(theta10)+l2*np.sin(theta20)])
y2s = deque([-l1*np.cos(theta10)-l2*np.cos(theta20)])
t_series = np.linspace(t_start, t_end, steps+1)
T1_series = deque([m1*(l1**2)*(w10**2)/2])
T2_series = deque([(m2*((l1**2)*(w10**2)+(l2**2)*(w20**2)
                  + 2*l1*l2*np.cos(theta10-theta20)*w10*w20))/2])
U1_series = deque([-m1*g*l1*np.cos(theta10)])
U2_series = deque([-m2*g*(l1*np.cos(theta10)+l2*np.cos(theta20))])
H_series = deque([T1_series[0]+T2_series[0]+U1_series[0]+U2_series[0]])

fig, ax = plt.subplots()
ax.set_xlabel('x␣/m', fontsize="14")
ax.set_ylabel('y␣/m', fontsize="14")
ax.grid()
fig.suptitle("$m_1={0},␣m_2={1},␣l_1={2},␣l_2={3}$".format(m1, m2, l1, l2),
             fontsize="20")
```

```


def f(theta1, theta2, w1, w2):
    phi = theta1 - theta2
    numerator = (-m2*l2*(w2**2)*np.sin(phi)
                 - (m1+m2)*g*np.sin(theta1)
                 - m2*l1*(w1**2)*np.sin(phi)*np.cos(phi)
                 + m2*g*np.cos(phi)*np.sin(theta2))
    denominator = (m1+m2)*l1-m2*l1*((np.cos(phi))**2)
    value = numerator/denominator
    return value


def h(theta1, theta2, w1, w2):
    phi = theta1-theta2
    numerator = (m2*l2*(w2**2)*np.cos(phi)*np.sin(phi)
                 + (m1+m2)*g*np.sin(theta1)*np.cos(phi)
                 + (m1+m2)*l1*(w1**2)*np.sin(phi)
                 - (m1+m2)*g*np.sin(theta2))
    denominator = (m1+m2)*l2 - m2*l2*((np.cos(phi)**2))
    value = numerator/denominator
    return value


def RungeKutta42(theta1, theta2, w1, w2, dt):
    k11 = f(theta1, theta2, w1, w2) # w1
    n11 = w1 # theta1
    k21 = h(theta1, theta2, w1, w2) # w2
    n21 = w2 # theta2

    k12 = f(theta1+n11*dt/2, theta2+n21*dt/2, w1+k11*dt/2, w2+k21*dt/2)
    n12 = w1 + k11*dt/2
    k22 = h(theta1+n11*dt/2, theta2+n21*dt/2, w1+k11*dt/2, w2+k21*dt/2)
    n22 = w2 + k21*dt/2

    k13 = f(theta1+n12*dt/2, theta2+n22*dt/2, w1+k12*dt/2, w2+k22*dt/2)
    n13 = w1 + k12*dt/2
    k23 = h(theta1+n12*dt/2, theta2+n22*dt/2, w1+k12*dt/2, w2+k22*dt/2)
    n23 = w2 + k22*dt/2

    k14 = f(theta1+n13*dt, theta2+n23*dt, w1+k13*dt, w2+k23*dt)
    n14 = w1 + k13*dt
    k24 = h(theta1+n13*dt, theta2+n23*dt, w1+k13*dt, w2+k23*dt)
    n24 = w2 + k23*dt

    w1 = w1 + (k11/6 + k12/3 + k13/3 + k14/6)*dt
    theta1 = theta1 + (n11/6 + n12/3 + n13/3 + n14/6)*dt

    w2 = w2 + (k21/6 + k22/3 + k23/3 + k24/6)*dt
```

```python
    theta2 = theta2 + (n21/6 + n22/3 + n23/3 + n24/6)*dt

    w1_series.append(w1)
    w2_series.append(w2)

    theta1_series.append(theta1)
    theta2_series.append(theta2)

    x1s.append(l1*np.sin(theta1))
    y1s.append(-l1*np.cos(theta1))

    x2s.append(l1*np.sin(theta1)+l2*np.sin(theta2))
    y2s.append(-l1*np.cos(theta1)-l2*np.cos(theta2))

    phi = theta1-theta2

    T1 = m1*(l1**2)*(w1**2)/2
    T2 = (m2*((l1**2)*(w1**2)+(l2**2)*(w2**2)
            + 2*l1*l2*np.cos(phi)*w1*w2))/2
    T1_series.append(T1)
    T2_series.append(T2)

    U1 = -m1*g*l1*np.cos(theta1)
    U2 = -m2*g*(l1*np.cos(theta1)+l2*np.cos(theta2))
    U1_series.append(U1)
    U2_series.append(U2)

    H = T1 + T2 + U1 + U2
    H_series.append(H)


for i in range(steps):
    theta1 = theta1_series[-1]
    theta2 = theta2_series[-1]
    w1 = w1_series[-1]
    w2 = w2_series[-1]
    RungeKutta42(theta1, theta2, w1, w2, dt)

images = []

for i in range(steps):
    x = [0, x1s[i], x2s[i]]
    y = [0, y1s[i], y2s[i]]

    image = ax.plot(x, y, 'o-', lw=2, c="black")
    ax.grid(True)
    ax.axis('equal')
    images.append(image)

```

```
ani = anim.ArtistAnimation(fig, images, interval=10)


dt_now = datetime.now()

ani.save('./figure/RK42/animation/{0}-{1}-{2}-{3}{4}{5}.gif'.format(
    dt_now.year, dt_now.month, dt_now.day,
    dt_now.hour, dt_now.minute, dt_now.second), writer='pillow', fps=50)
# plt.show()
```

ソースコード 6　二重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 = 1.0
l2 = 1.0
m1 = 1.0
m2 = 1.0
theta10 = np.pi/6
theta20 = np.pi/2
theta10_str = "pi6"
theta20_str = "pi2"
w10 = 0.0
w20 = 0.0
t_start = 0
t_end = 4
steps = 400

# calculated
dt = (t_end - t_start)/steps

# lists
theta1_series = deque([theta10])
theta2_series = deque([theta20])
w1_series = deque([w10])
w2_series = deque([w20])
x1s = deque([l1*np.sin(theta10)])
y1s = deque([-l1*np.cos(theta10)])
x2s = deque([l1*np.sin(theta10)+l2*np.sin(theta20)])
y2s = deque([-l1*np.cos(theta10)-l2*np.cos(theta20)])
t_series = np.linspace(t_start, t_end, steps+1)
T1_series = deque([m1*(l1**2)*(w10**2)/2])
T2_series = deque([(m2*((l1**2)*(w10**2)+(l2**2)*(w20**2)
                  + 2*l1*l2*np.cos(theta10-theta20)*w10*w20))/2])
U1_series = deque([-m1*g*l1*np.cos(theta10)])
U2_series = deque([-m2*g*(l1*np.cos(theta10)+l2*np.cos(theta20))])
H_series = deque([T1_series[0]+T2_series[0]+U1_series[0]+U2_series[0]])


fig = plt.figure()
ax1 = fig.add_subplot(221)
```

```
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(223)
ax4 = fig.add_subplot(224)

ax1.set_xlabel("time␣/s")
ax1.set_ylabel("energy␣/J")
ax2.set_xlabel("time␣/s")
ax2.set_ylabel("energy␣/J")
ax3.set_xlabel("time␣/s")
ax3.set_ylabel("energy␣/J")
ax4.set_xlabel("time␣/s")
ax4.set_ylabel(R"$\log|1-E/E_0|$")


def f(theta1, theta2, w1, w2):
    phi = theta1 - theta2
    numerator = (-m2*l2*(w2**2)*np.sin(phi)
                 - (m1+m2)*g*np.sin(theta1)
                 - m2*l1*(w1**2)*np.sin(phi)*np.cos(phi)
                 + m2*g*np.cos(phi)*np.sin(theta2))
    denominator = (m1+m2)*l1-m2*l1*((np.cos(phi))**2)
    value = numerator/denominator
    return value


def h(theta1, theta2, w1, w2):
    phi = theta1-theta2
    numerator = (m2*l2*(w2**2)*np.cos(phi)*np.sin(phi)
                 + (m1+m2)*g*np.sin(theta1)*np.cos(phi)
                 + (m1+m2)*l1*(w1**2)*np.sin(phi)
                 - (m1+m2)*g*np.sin(theta2))
    denominator = (m1+m2)*l2 - m2*l2*((np.cos(phi)**2))
    value = numerator/denominator
    return value


def RungeKutta42(theta1, theta2, w1, w2, dt):
    k11 = f(theta1, theta2, w1, w2) # w1
    n11 = w1 # theta1
    k21 = h(theta1, theta2, w1, w2) # w2
    n21 = w2 # theta2

    k12 = f(theta1+n11*dt/2, theta2+n21*dt/2, w1+k11*dt/2, w2+k21*dt/2)
    n12 = w1 + k11*dt/2
    k22 = h(theta1+n11*dt/2, theta2+n21*dt/2, w1+k11*dt/2, w2+k21*dt/2)
    n22 = w2 + k21*dt/2

    k13 = f(theta1+n12*dt/2, theta2+n22*dt/2, w1+k12*dt/2, w2+k22*dt/2)
    n13 = w1 + k12*dt/2
```

```
    k23 = h(theta1+n12*dt/2, theta2+n22*dt/2, w1+k12*dt/2, w2+k22*dt/2)
    n23 = w2 + k22*dt/2

    k14 = f(theta1+n13*dt, theta2+n23*dt, w1+k13*dt, w2+k23*dt)
    n14 = w1 + k13*dt
    k24 = h(theta1+n13*dt, theta2+n23*dt, w1+k13*dt, w2+k23*dt)
    n24 = w2 + k23*dt

    w1 = w1 + (k11/6 + k12/3 + k13/3 + k14/6)*dt
    theta1 = theta1 + (n11/6 + n12/3 + n13/3 + n14/6)*dt

    w2 = w2 + (k21/6 + k22/3 + k23/3 + k24/6)*dt
    theta2 = theta2 + (n21/6 + n22/3 + n23/3 + n24/6)*dt

    w1_series.append(w1)
    w2_series.append(w2)

    theta1_series.append(theta1)
    theta2_series.append(theta2)

    x1s.append(l1*np.sin(theta1))
    y1s.append(-l1*np.cos(theta1))

    x2s.append(l1*np.sin(theta1)+l2*np.sin(theta2))
    y2s.append(-l1*np.cos(theta1)-l2*np.cos(theta2))

    phi = theta1-theta2

    T1 = m1*(l1**2)*(w1**2)/2
    T2 = (m2*((l1**2)*(w1**2)+(l2**2)*(w2**2)
             + 2*l1*l2*np.cos(phi)*w1*w2))/2
    T1_series.append(T1)
    T2_series.append(T2)

    U1 = -m1*g*l1*np.cos(theta1)
    U2 = -m2*g*(l1*np.cos(theta1)+l2*np.cos(theta2))
    U1_series.append(U1)
    U2_series.append(U2)

    H = T1 + T2 + U1 + U2
    H_series.append(H)


for i in range(steps):
    theta1 = theta1_series[-1]
    theta2 = theta2_series[-1]
    w1 = w1_series[-1]
    w2 = w2_series[-1]
    RungeKutta42(theta1, theta2, w1, w2, dt)
```

```
H_series = np.array(H_series)
T1_series = np.array(T1_series)
T2_series = np.array(T2_series)
U1_series = np.array(U1_series)
U2_series = np.array(U2_series)
H1_series = T1_series + U1_series
H2_series = T2_series + U2_series

ax1.plot(t_series, H_series, label="total␣energy")
ax1.grid()
ax1.legend()

ax2.plot(t_series, T1_series, label="$T_1$")
ax2.plot(t_series, U1_series, label="$U_1$")
ax2.plot(t_series, T2_series, label="$T_2$")
ax2.plot(t_series, U2_series, label="$U_2$")
ax2.grid()
ax2.legend()

ax3.plot(t_series, H1_series, label="$H_1$")
ax3.plot(t_series, H2_series, label="$H_2$")
ax3.grid()
ax3.legend()

exact_H = H_series[0]
print(exact_H)
logH_series = np.log(np.abs(1-H_series/exact_H))
ax4.plot(t_series, logH_series)
ax4.grid()

plt.tight_layout()

fig.savefig('./figure/RK42/evaluation2/{6}_{7}_{0}-{1}-{2}-{3}{4}{5}.jpeg'
            .format(dt_now.year, dt_now.month, dt_now.day, dt_now.hour,
                    dt_now.minute, dt_now.second,
                    theta10_str, theta20_str))
```

ソースコード 7　二重振り子における 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 = 1.0
l2 = 1.0
m1 = 1.0
m2 = 1.0
w10 = 0.0
w20 = 0.0
t_start = 0
t_end = 4


# lists
T1_series = deque([])
T2_series = deque([])
U1_series = deque([])
U2_series = deque([])
H_series = deque([])


def f(theta1, theta2, w1, w2):
    phi = (theta1 - theta2) % (2*np.pi)
    numerator = (-m2*l2*(w2**2)*np.sin(phi)
                 - (m1+m2)*g*np.sin(theta1)
                 - m2*l1*(w1**2)*np.sin(phi)*np.cos(phi)
                 + m2*g*np.cos(phi)*np.sin(theta2))
    denominator = (m1+m2)*l1-m2*l1*((np.cos(phi))**2)
    value = numerator/denominator
    return value


def h(theta1, theta2, w1, w2):
    phi = (theta1-theta2) % (2*np.pi)
    numerator = (m2*l2*(w2**2)*np.cos(phi)*np.sin(phi)
                 + (m1+m2)*g*np.sin(theta1)*np.cos(phi)
                 + (m1+m2)*l1*(w1**2)*np.sin(phi)
                 - (m1+m2)*g*np.sin(theta2))
    denominator = (m1+m2)*l2 - m2*l2*((np.cos(phi)**2))
    value = numerator/denominator
    return value
```

```


def RungeKutta42(theta1, theta2, w1, w2, dt):
    k11 = f(theta1, theta2, w1, w2) # w1
    n11 = w1 # theta1
    k21 = h(theta1, theta2, w1, w2) # w2
    n21 = w2 # theta2

    k12 = f(theta1+n11*dt/2, theta2+n21*dt/2, w1+k11*dt/2, w2+k21*dt/2)
    n12 = w1 + k11*dt/2
    k22 = h(theta1+n11*dt/2, theta2+n21*dt/2, w1+k11*dt/2, w2+k21*dt/2)
    n22 = w2 + k21*dt/2

    k13 = f(theta1+n12*dt/2, theta2+n22*dt/2, w1+k12*dt/2, w2+k22*dt/2)
    n13 = w1 + k12*dt/2
    k23 = h(theta1+n12*dt/2, theta2+n22*dt/2, w1+k12*dt/2, w2+k22*dt/2)
    n23 = w2 + k22*dt/2

    k14 = f(theta1+n13*dt, theta2+n23*dt, w1+k13*dt, w2+k23*dt)
    n14 = w1 + k13*dt
    k24 = h(theta1+n13*dt, theta2+n23*dt, w1+k13*dt, w2+k23*dt)
    n24 = w2 + k23*dt

    w1 = w1 + (k11/6 + k12/3 + k13/3 + k14/6)*dt
    theta1 = (theta1 + (n11/6 + n12/3 + n13/3 + n14/6)*dt) % (2*np.pi)

    w2 = w2 + (k21/6 + k22/3 + k23/3 + k24/6)*dt
    theta2 = (theta2 + (n21/6 + n22/3 + n23/3 + n24/6)*dt) % (2*np.pi)

    w1_series.append(w1)
    w2_series.append(w2)

    theta1_series.append(theta1)
    theta2_series.append(theta2)

    phi = (theta1-theta2) % (2*np.pi)

    T1 = m1*(l1**2)*(w1**2)/2
    T2 = (m2*((l1**2)*(w1**2)+(l2**2)*(w2**2)
             + 2*l1*l2*np.cos(phi)*w1*w2))/2
    T1_series.append(T1)
    T2_series.append(T2)

    U1 = -m1*g*l1*np.cos(theta1)
    U2 = -m2*g*(l1*np.cos(theta1)+l2*np.cos(theta2))
    U1_series.append(U1)
    U2_series.append(U2)

    H = T1 + T2 + U1 + U2
```

```
    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"]
# thetas = [np.pi/12, np.pi/24]
# thetas_str = ["pi12", "pi24"]
# thetas_tex = [R"\pi/12", R"\pi/24"]
dts = np.logspace(-5, -2, num=7, base=10.0)


for i in range(len(thetas)):
    for j in range(len(thetas)):
        theta10 = thetas[i]
        theta20 = thetas[j]
        theta10_str = thetas_str[i]
        theta20_str = thetas_str[j]
        theta10_tex = thetas_tex[i]
        theta20_tex = thetas_tex[j]

        fig, ax = plt.subplots()

        phi0 = (theta10-theta20) % (2*np.pi)

        T10 = m1*(l1**2)*(w10**2)/2
        T20 = (m2*((l1**2)*(w10**2)+(l2**2)*(w20**2)
                  + 2*l1*l2*np.cos(phi0)*w10*w20))/2
        U10 = -m1*g*l1*np.cos(theta10)
        U20 = -m2*g*(l1*np.cos(theta10)+l2*np.cos(theta20))
        exact_H = T10 + T20 + U10 + U20

        y = []

        for dt in dts:
            steps = int((t_end - t_start) / dt)
            theta1_series = deque([theta10])
            theta2_series = deque([theta20])
            w1_series = deque([w10])
            w2_series = deque([w20])

            for step in range(steps):
                theta1 = theta1_series[-1]
                theta2 = theta2_series[-1]
                w1 = w1_series[-1]
                w2 = w2_series[-1]
                RungeKutta42(theta1, theta2, w1, w2, dt)

            # y.append(np.log(np.abs(1 - H_series[-1]/exact_H)))
```

```
            y.append(np.abs(H_series[-1]-exact_H)) # when L2 norm

        ax.plot(dts, y, label=R"$\theta_1 = {0}, \theta_2 = {1}$"
                .format(theta10_tex, theta20_tex))
        ax.semilogx()
        ax.semilogy() # when L2 norm
        ax.grid()
        ax.legend(fontsize="14")
        ax.set_xlabel(R"$ \Delta t$", fontsize="14")
        ax.set_ylabel(R"$|E-E_0|$", fontsize="14") # when L2 norm
        # ax.set_ylabel(R"$\log|1-E/E_0|$")
        ax.set_title('Evaluation of RK4 (Double Pendulum)', fontsize="16")
        fig.tight_layout()
        fig.savefig('./figure/RK42/evaluation1/{6}_{7}_{0}-{1}-{2}-{3}{4}{5}.jpeg'
                    .format(dt_now.year, dt_now.month, dt_now.day, dt_now.hour,
                            dt_now.minute, dt_now.second,
                            theta10_str, theta20_str))
```

4.4 参考文献

- 畑浩之 「基幹講座 物理学 解析力学」 東京図書