Simulation of Large Angle Pendulum by RK4(4th Order Runge-Kutta Method)

B13902077

資工一 李承翰

I. Introduction

This is a project about using RK4 method to simulate large angle pendulum. In this document, only parts of the full code will be shown, most repetitive contents and minor codes for format arrangement and figure making will be ignored. The full code will be on this GitHub link.

II. 4th Order Runge-Kutta Method

Runge-Kutta method is a way to approximate solutions of simultaneous nonlinear equations. With the help of computers, 4th order Runge-Kutta method, one of the Runge-Kutta methods, can be easily applied to equations when initial condition is known.

With the initial condition:

$$y' = f(t, y), y(t_0) = y_0$$

The whole solution can be derived using RK4, by calculating:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Where

$$k_1 = f(t_n, y_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right)$$

$$k_4 = f(t_n + h, y_n + hk_3)$$

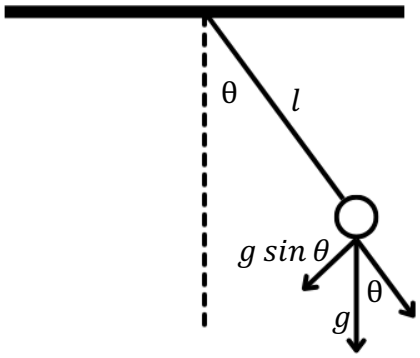
And h is a tiny time interval.

When it comes to solve a pendulum problem, we usually assume the angle is small, so that we can use the approximate $\lim_{\theta \to 0} \frac{\sin \theta}{\theta} = 1$ to get an easier solution. By comparison, large angle pendulum motions are way more complex. Hence, we are going to use RK4 method to simulate its equation of motion. We will apply RK4 method to get the graph of $\theta(t)$ and $\omega(t)$, under the condition of different initial angular velocities, initial angular accelerations and pendulum lengths.

III. Large Angle Pendulum with No Damping Effect

a. Apply Question to RK4

We first define the parameters and analyze the forces.



In order to use RK4 method to approximate the equation of motion, we need to figure out the derivative of θ and ω.

$$\frac{d}{dt}\theta = \omega$$

From the force analysis, we can get the angular acceleration, which is the derivative of ω.

$$\frac{d}{dt}\omega = \frac{-g \sin \theta}{l}$$

b. Plotting with Python

Again, the full code of the project has been posted on this GitHub link. Some repetitive codes or functional codes won't be explained and showed in this document.

The derivatives of θ and ω can be obtained using the `derivation` function.

```
# function of derivation (doing theta and omega at once)
def derivation(theta, omega, t, l):
    dtheta_dh = omega
    domega_dh = -g * np.sin(theta) / l
    return dtheta_dh, domega_dh
```

Then, with the derivation function, we can complete the `rk4` function. In order to check if our code is functioning, we print out theta and omega.

```
#function of RK4
def rk4(theta, omega, t, h, l):
    print(theta, omega)
    # calculation of k1 k2 k3 k4
    k1_theta, k1_omega = derivation(theta, omega, t, l)
    k2_theta, k2_omega = derivation(theta + 0.5 * k1_theta * h, omega +
0.5 * k1_omega * h, t + 0.5 * h, l)
    k3_theta, k3_omega = derivation(theta + 0.5 * k2_theta * h, omega +
0.5 * k2_omega * h, t + 0.5 * h, l)
    k4_theta, k4_omega = derivation(theta + k3_theta * h, omega +
k3_omega * h, t + h, l)

    # calculation of result
    next_theta = theta + (h / 6) * (k1_theta + 2 * k2_theta + 2 *
k3_theta + k4_theta)
    next_omega = omega + (h / 6) * (k1_omega + 2 * k2_omega + 2 *
k3_omega + k4_omega)
    print(next_theta, next_omega)
    return next_theta, next_omega
```

First, the control variable is the initial θ, vary from $15°, 30°, 45°, \ldots, 165°$. Also, we set $h = 0.0001(\text{sec})$. Additionally, the when ω changes between positive and negative for the second time (from negative to positive, then from positive to negative), that means it has completed a period of motion. Finally, after we get the series of θ, ω and t, we can plot all the combinations of datums.

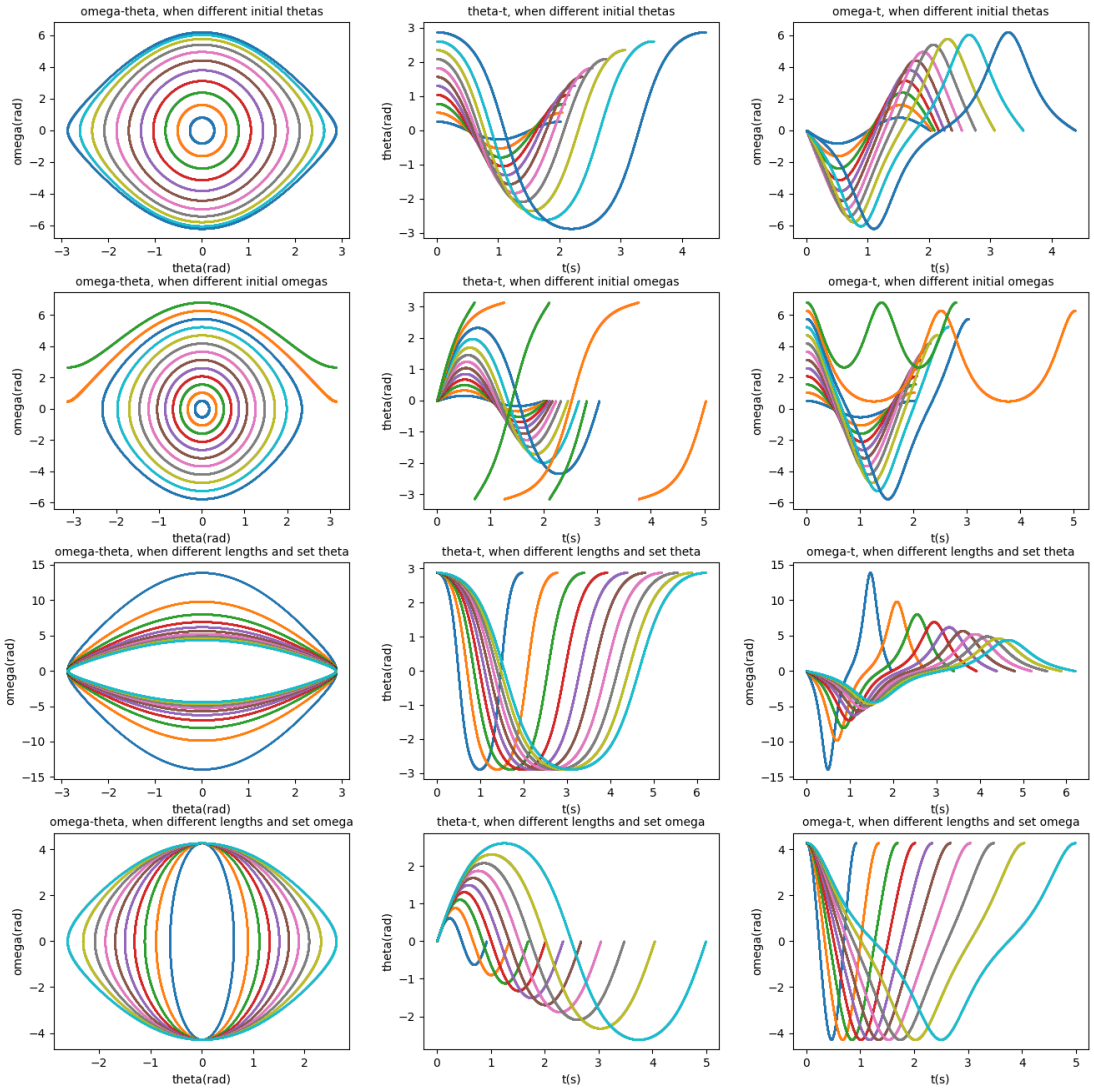
```
#control variable: initial theta (from 15 degree to 165 degree, step = 15
degree)
#independent variables: initial omega = 0, l = 1
initial_theta = np.radians(15)
initial_omega = 0
theta_step = np.radians(15)
l = 1
while (initial_theta <= np.radians(166)):
    ts = [0]
    h = 0.0001
    thetas = [initial_theta]
    omegas = [initial_omega]

    #set ending condition: when omega changes between positive and
negative for the second time
    count = 0
    while (count < 2):
        t_next = ts[-1] + h
        theta_next, omega_next = rk4(thetas[-1], omegas[-1], ts[-1], h,
l)
        if (omegas[-1] * omega_next < 0):
            count += 1
        ts.append(t_next)
        thetas.append(theta_next)
        omegas.append(omega_next)
    omega_theta_initial_theta.plot(thetas, omegas, linewidth = 1,
linestyle = "None", marker = ".", markersize = 0.5)
    theta_t_initial_theta.plot(ts, thetas, linewidth = 1, linestyle =
"None", marker = ".", markersize = 0.5)
    omega_t_initial_theta.plot(ts, omegas, linewidth = 1, linestyle =
"None", marker = ".", markersize = 0.5)
```

Similarly, we can get the datums of different ω, different l with initial $\theta = 0$ and different l with initial $\omega = 0$. Notably, when the initial $\theta = 0$, the ending condition is when θ changes between positive and negative for the second time. What's more, while the initial ω is big enough, there will be a vertical circular motion. In order to

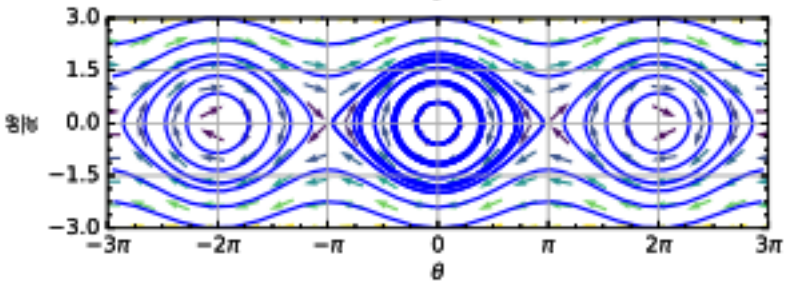
maintain the size of the axis of θ, we'll subtract 2π from θ whenever it's greater than π and add 2π whenever it's lower than $-\pi$. Here is the modified code for the above initial condition.

```
#control variable: initial omega (from 30 degree/s to 390 degree/s, step
= 30 degree/s)
#independent variables: initial theta = 0, l = 1
initial_theta = 0
initial_omega = np.radians(30)
omega_step = np.radians(30)
l = 1
while (initial_omega <= np.radians(391)):
    ts = [0]
    h = 0.0001
    thetas = [initial_theta]
    omegas = [initial_omega]

    #set ending condition: when theta changes between positive and
negative for the second time
    count = 0
    while (count < 2):
        t_next = ts[-1] + h
        theta_next, omega_next = rk4(thetas[-1], omegas[-1], ts[-1], h,
l)
        if (thetas[-1] * theta_next < 0):
            count += 1
        if (theta_next < np.radians(-180)):
            theta_next += np.radians(360)
        if (theta_next > np.radians(180)):
            theta_next -= np.radians(360)
        ts.append(t_next)
        thetas.append(theta_next)
        omegas.append(omega_next)
    omega_theta_initial_omega.plot(thetas, omegas, linewidth = 1,
linestyle = "None", marker = ".", markersize = 0.5)
    theta_t_initial_omega.plot(ts, thetas, linewidth = 1, linestyle =
"None", marker = ".", markersize = 0.5)
    omega_t_initial_omega.plot(ts, omegas, linewidth = 1, linestyle =
"None", marker = ".", markersize = 0.5)
```

After some arrangement and format adjusting, here is the result.



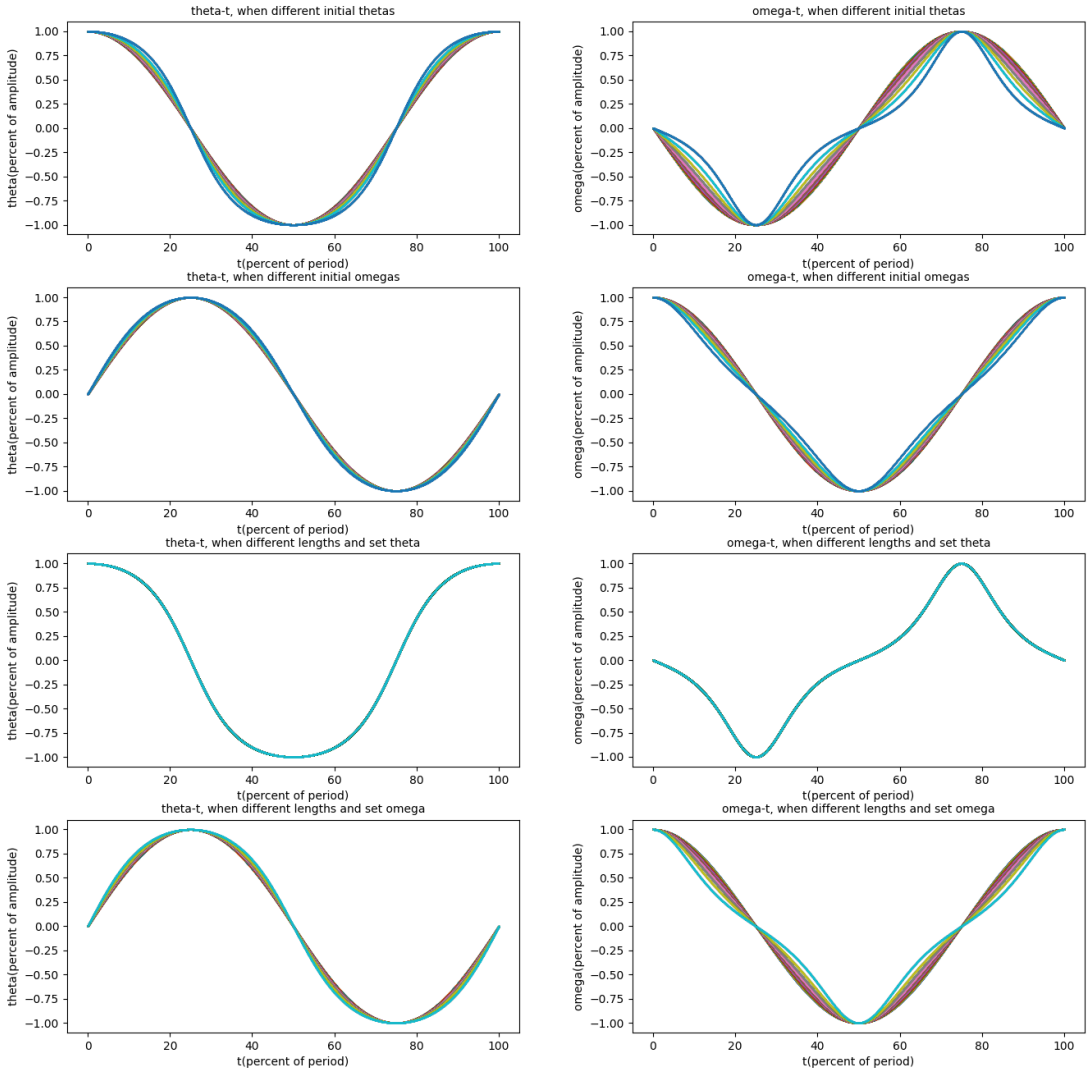
And here is the graph of phase space on Wikipedia, which is really similar with my simulation result.



Some interesting findings are:

- The period is positively related to the initial θ, the initial ω and the length of the pendulum, while small angle pendulum periods is irrelated to the initial θ and the initial ω.
- The two open lines on the $\omega - \theta$ graph of different initial ω represents the vertical circular motion. This is also the reason why there are two datums acting different from others in the next two graphs.

In order to better observe the results, we perform normalization (Min-Max Scaling). Thus, we resize each data set from $\theta - t$ and $\omega - t$ graphs. θ and ω are divided by their max value, and t is divided by the period. The cases of vertical circular motion are ignored here.

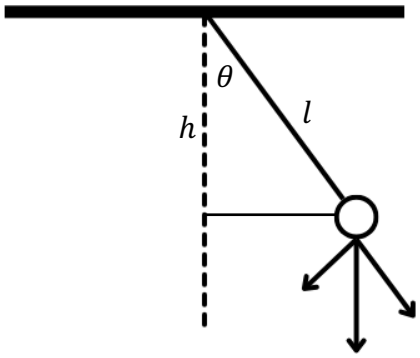


In the above graphs, we can learn that:

- The deviation between the equation of motion and regular sine waves is larger when initial θ and ω is larger. It tends to have more sub-hills.
- As we set initial θ and change lengths, the equation of motion fits with each other after resizing. That is, the initial θ is the key factor of swinging mode.

c. Find Accurate Period and Equation of Motion

We can write out the differential equation from the energy perspective.



Let initial angle = θ_0, then the potential energy the bob releases is:

$$mg(h - h_0) = mgl(\cos\theta - \cos\theta_0)$$

Meanwhile, the kinetic energy:

$$\frac{1}{2}mv^2 = \frac{1}{2}ml^2\left(\frac{d}{dt}\theta\right)^2$$

Since there's no energy lost in this system,

$$\frac{1}{2}ml^2\left(\frac{d}{dt}\theta\right)^2 = mgl(\cos\theta - \cos\theta_0)$$

So,

$$\frac{d}{dt}\theta = \sqrt{\frac{2g}{l}(\cos\theta - \cos\theta_0)}$$

And

$$\frac{d}{d\theta}t = \sqrt{\frac{l}{2g}}\frac{1}{\sqrt{\cos\theta - \cos\theta_0}}$$

A full period is 4 times the time that θ travels from 0 to θ_0, so

$$T = 4\sqrt{\frac{l}{2g}}\int_0^{\theta_0}\frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}}$$

$$\int_0^{\theta_0}\frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}} = \int_0^{\theta_0}\frac{d\theta}{\sqrt{2\sin^2\frac{\theta_0}{2} - 2\sin^2\frac{\theta}{2}}} = \sqrt{\frac{1}{2}}\int_0^{\theta_0}\frac{\frac{1}{\sin\frac{\theta_0}{2}}d\theta}{\sqrt{1 - \frac{\sin^2\frac{\theta}{2}}{\sin^2\frac{\theta_0}{2}}}}$$

Let $\sin u = \frac{\sin\frac{\theta}{2}}{\sin\frac{\theta_0}{2}} \Rightarrow \cos u\, du = \frac{1}{2\sin\frac{\theta_0}{2}}\cos\frac{\theta}{2}d\theta$

Let $k = \sin\frac{\theta_0}{2} \Rightarrow k\sin u = \sin\frac{\theta}{2}$

Then,

$$\int_0^{\theta_0}\frac{\frac{1}{\sin\frac{\theta_0}{2}}d\theta}{\sqrt{1 - \frac{\sin^2\frac{\theta}{2}}{\sin^2\frac{\theta_0}{2}}}} = \int_{u=0}^{u=\frac{\pi}{2}}\frac{\frac{2\cos u}{\cos\frac{\theta}{2}}du}{\sqrt{1-\sin^2 u}} = 2\int_{u=0}^{u=\frac{\pi}{2}}\frac{du}{\cos\frac{\theta}{2}} = 2\int_0^{\frac{\pi}{2}}\frac{du}{\sqrt{1-k^2\sin^2 u}}$$

Which is in the form of complete elliptic integral of the first kind.

$$K(k) = \int_0^{\frac{\pi}{2}}\frac{du}{\sqrt{1-k^2\sin^2 u}}$$

The solution of $K(k)$ is:

$$K(k) = \frac{\pi}{2}\sum_{n=0}^{\infty}\left(\frac{(2n-1)!!}{(2n)!!}k^n\right)^2$$

$$= \frac{\pi}{2}\left(1 + \left(\frac{1}{2}\right)^2 k^2 + \left(\frac{1\times 3}{2\times 4}\right)^2 k^4 + \left(\frac{1\times 3\times 5}{2\times 4\times 6}\right)^2 k^6 + \ldots\right)$$

So, we can obtain the period

$$T = 2\pi\sqrt{\frac{l}{g}}\left(1 + \left(\frac{1}{2}\right)^2\sin^2\frac{\theta_0}{2} + \left(\frac{1\times 3}{2\times 4}\right)^2\sin^4\frac{\theta_0}{2} + \left(\frac{1\times 3\times 5}{2\times 4\times 6}\right)^2\sin^6\frac{\theta_0}{2} + \ldots\right)$$

Then, we can calculate in our codes and compare it with the simulation result.

```
def calculatePeriod(theta0, h, l, g):
    k = np.sin(theta0/2)
    sum = 0
    an = 1 * 2 * 3.1415926 * np.sqrt(l/g) #the first element
    n = 0
    while an > 0.1 * h:
        #end when an is smaller than 0.1 * h since h is the minimum time
span of our simulation
        sum += an
        n += 1
        an = an * ((2 * n - 1) / (2 * n) * k) ** 2 #calculate a(n+1) with
an
    return sum
```

We'll print out everything in the end so that we can compare at once.

```
angle_period = []
while (initial_theta <= np.radians(166)):
    ts = [0]
    h = 0.0001
    thetas = [initial_theta]
    omegas = [initial_omega]

    #set                                        : when omega changes
between positive and negative for the second time
    count = 0
    while (count < 2):
        t_next = ts[-1] + h
        theta_next, omega_next = rk4(thetas[-1], omegas[-1], ts[-1], h, l)
        if (omegas[-1] * omega_next < 0):
            count += 1
        ts.append(t_next)
        thetas.append(theta_next)
        omegas.append(omega_next)
    angle_period.append(
        (round(np.rad2deg(initial_theta)),
         "%.4f"%calculatePeriod(initial_theta, h, l, g),
         "%.4f"%ts[-1]))
    initial_theta += theta_step

for theta_init, calc_period, simu_period in angle_period:
    print(f"initial theta = {theta_init}:\ncalculated period is
{calc_period}, simulated period is {simu_period}")
```

And the result:

```
initial theta = 15 degree:
calculated period is 2.0147, simulated period is 2.0147
initial theta = 30 degree:
calculated period is 2.0410, simulated period is 2.0410
initial theta = 45 degree:
calculated period is 2.0862, simulated period is 2.0863
initial theta = 60 degree:
calculated period is 2.1529, simulated period is 2.1529
initial theta = 75 degree:
calculated period is 2.2447, simulated period is 2.2448
initial theta = 90 degree:
calculated period is 2.3678, simulated period is 2.3679
initial theta = 105 degree:
calculated period is 2.5321, simulated period is 2.5321
initial theta = 120 degree:
calculated period is 2.7541, simulated period is 2.7541
initial theta = 135 degree:
calculated period is 3.0651, simulated period is 3.0652
initial theta = 150 degree:
calculated period is 3.5350, simulated period is 3.5351
initial theta = 165 degree:
calculated period is 4.3836, simulated period is 4.3842
```

The calculated period and the simulated period is almost the same, the deviation between them can be resulted from:

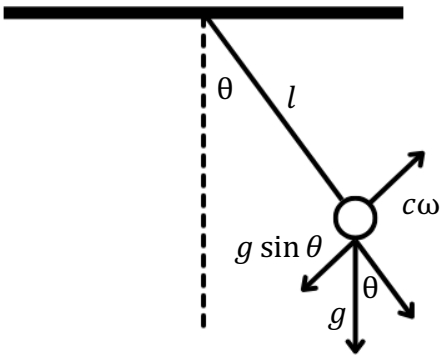
- Floating error in codes
- Error of RK4 method
- Error of sumation of the series

But still, all the error rate between two periods is lower than 0.1%, so we can see these as successful simulations.

IV. Large Angle Pendulum with Damping Effect

a. Apply Question to RK4

First, analyze the forces.



We set the damping coefficient as c, which means the angular acceleration caused by damping effect is,

$$\alpha_{\text{damping}} = -c\omega$$

Then, we can find the derivatives of θ and ω.

$$\frac{d}{dt}\theta = \omega$$

$$\frac{d}{dt}\omega = \frac{-g\sin\theta}{l} - c\omega$$

b. Plotting with Python

We want to focus on the effect of damping here, so the control variables will be initial angles under different camping coefficients. Still, three graphs of $\omega - \theta$, $\theta - t$ and $\omega - t$ will be drawn.

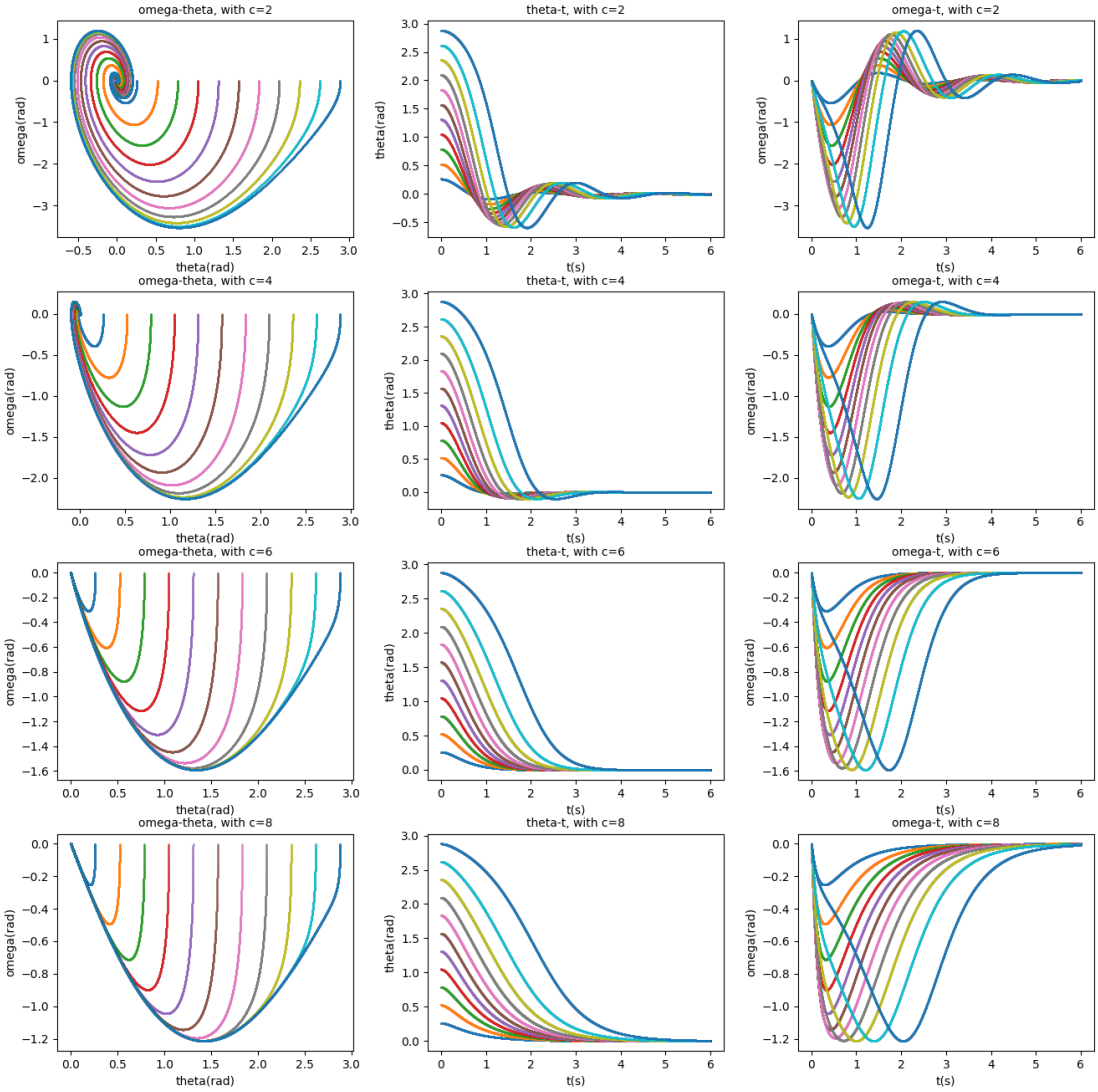
We can adjust the codes by rewrite the `derivation` function only. The codes of `rk4` function and other parts are almost the same.

```
# function of derivation (doing theta and omega at once)
def derivation(theta, omega, t, l, c):
    dtheta_dh = omega
    domega_dh = -g * np.sin(theta) / l - c * omega
    return dtheta_dh, domega_dh
```

The initial angles are $15°, 30°, 45°, \ldots, 165°$. The damping coefficients are $2, 4, 6, 8$. After several test runs, we decided that the ending condition is when time has reached 6s. The following code for is the case $c = 2$.

```
#control variable: initial theta (from 15 degree to 165 degree, step = 15
degree)
#independent variables: initial omega = 0, l = 1, c = 1
initial_theta = np.radians(15)
initial_omega = 0
theta_step = np.radians(15)
l = 1
c = 2
while (initial_theta <= np.radians(166)):
    ts = [0]
    h = 0.0001
    thetas = [initial_theta]
    omegas = [initial_omega]

    #set ending condition: when t = 6
    while (ts[-1] <= 6):
        t_next = ts[-1] + h
        theta_next, omega_next = rk4(thetas[-1], omegas[-1], ts[-1], h,
l, c)
        ts.append(t_next)
        thetas.append(theta_next)
        omegas.append(omega_next)
    omega_theta_c1.plot(thetas, omegas, linewidth = 1, linestyle =
"None", marker = ".", markersize = 0.2)
    theta_t_c1.plot(ts, thetas, linewidth = 1, linestyle = "None", marker
= ".", markersize = 0.2)
    omega_t_c1.plot(ts, omegas, linewidth = 1, linestyle = "None", marker
= ".", markersize = 0.2)
    initial_theta += theta_step
```

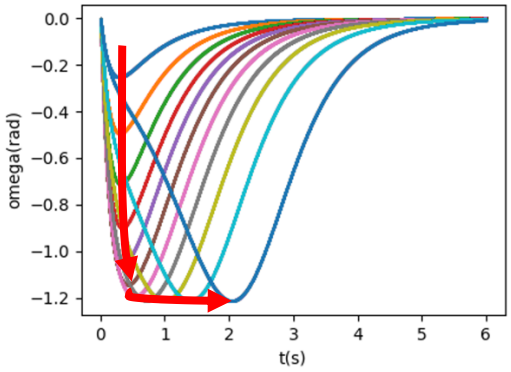
This is the result.



From the simulation, we can observe that:

- As the damping coefficient increases, the swinging mode of large angle pendulum changes from underdamping to overdamping just like small angle pendulums.
- When the initial θ increases, the magnitude of peak ω increases and the time it takes to reach peak stays at first. Then, the magnitude of peak ω stays and

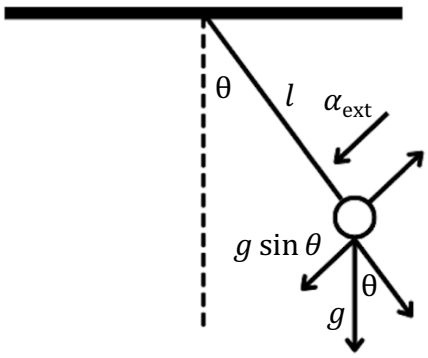
the time it takes to reach peak increases. (like the following graph)



V. Forced Large Angle Pendulum with Damping Effect

a. Apply Question to RK4

First, analyze the forces.



α_{ext} is the angular acceleration caused by external force. Here, we assume

$$\alpha_{\text{ext}} = \alpha_{\max} \sin \omega_\alpha t$$

Then, we can rewrite the derivatives of θ and ω.

$$\frac{d}{dt}\theta = \omega$$

$$\frac{d}{dt}\omega = \frac{-g \sin\theta}{l} - c\omega + \alpha_{\text{ext}} = \frac{-g \sin\theta}{l} - c\omega + \alpha_{\max} \sin \omega_\alpha t$$

b. Plotting with Python

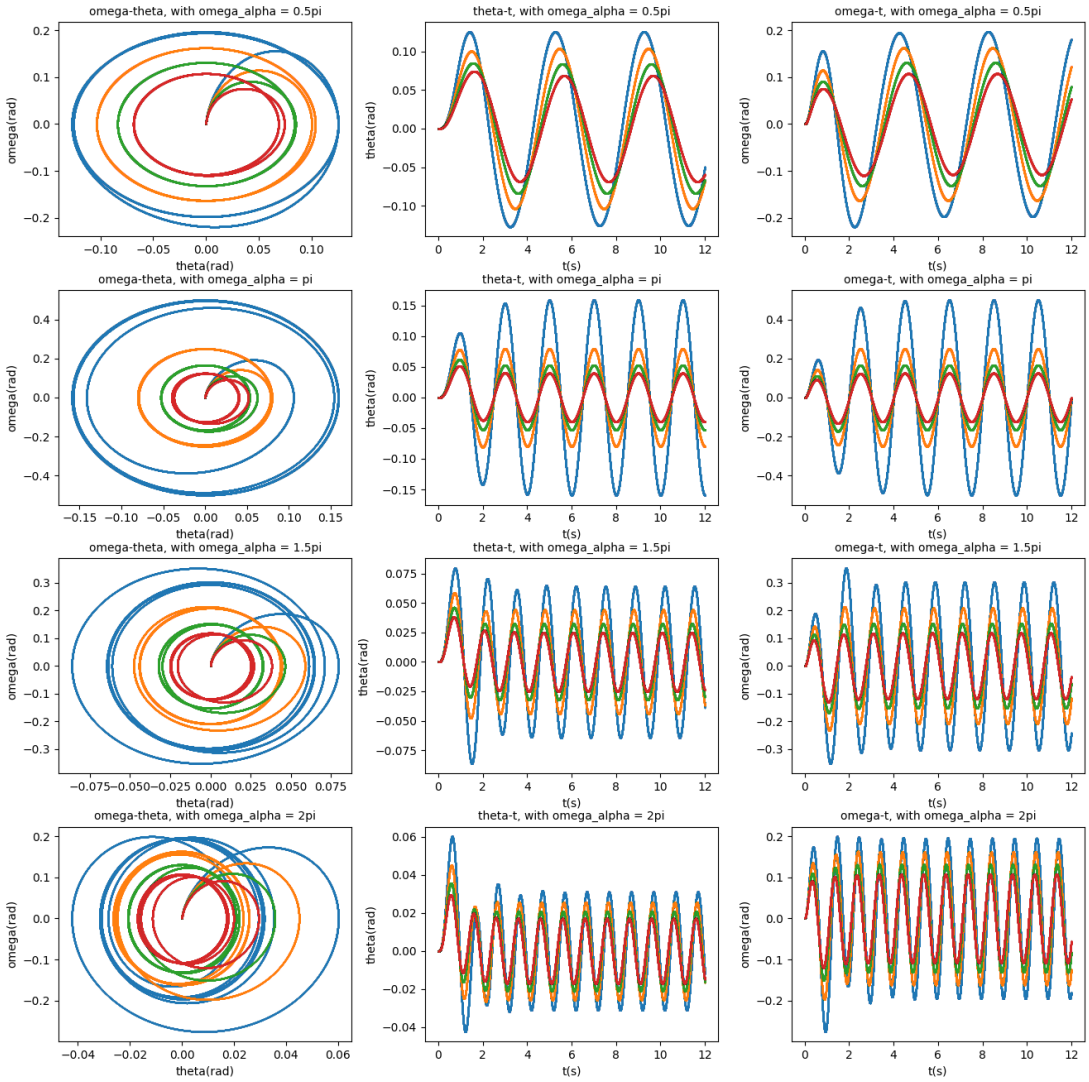
After adding the external angular acceleration into the problem, here is the code of `derivation` and `rk4` function.

```
# function of derivation (doing theta and omega at once)
def derivation(theta, omega, t, l, c, alpha_max, omega_alpha):
    dtheta_dh = omega
    domega_dh = -g * np.sin(theta) / l - c * omega + alpha_max *
np.sin(omega_alpha * t)
    return dtheta_dh, domega_dh

#function of RK4
def rk4(theta, omega, t, h, l, c, alpha_max, omega_alpha):
    print(theta, omega)
    # calculation of k1 k2 k3 k4
    k1_theta, k1_omega = derivation(theta, omega, t, l, c, alpha_max,
omega_alpha)
    k2_theta, k2_omega = derivation(theta + 0.5 * k1_theta * h, omega +
0.5 * k1_omega * h, t + 0.5 * h, l, c, alpha_max, omega_alpha)
    k3_theta, k3_omega = derivation(theta + 0.5 * k2_theta * h, omega +
0.5 * k2_omega * h, t + 0.5 * h, l, c, alpha_max, omega_alpha)
    k4_theta, k4_omega = derivation(theta + k3_theta * h, omega +
k3_omega * h, t + h, l, c, alpha_max, omega_alpha)

    # calculation of result
    next_theta = theta + (h / 6) * (k1_theta + 2 * k2_theta + 2 *
k3_theta + k4_theta)
    next_omega = omega + (h / 6) * (k1_omega + 2 * k2_omega + 2 *
k3_omega + k4_omega)
    print(next_theta, next_omega)
    return next_theta, next_omega
```

We set the initial θ and ω to 0, l to 1 and $\alpha_{\max}$ to 1, the control variable will be the damping coefficient c under different ω_α. The following is the code for $\omega_\alpha = 0.5\pi$.

```
#control variable: c (from 2 to 8, step = 2)
#independent variables: initial theta = 0, initial omega = 0, l = 1,
alpha_max = 1, omega_alpha = 0.5pi
initial_theta = 0
initial_omega = 0
l = 1
c = 2
c_step = 2
alpha_max = 1
omega_alpha = 0.5 * 3.1415926
while (c <= 8):
    ts = [0]
    h = 0.0001
    thetas = [initial_theta]
    omegas = [initial_omega]

    #set ending condition: when t = 12
    while (ts[-1] <= 12):
        t_next = ts[-1] + h
        theta_next, omega_next = rk4(thetas[-1], omegas[-1], ts[-1], h,
l, c, alpha_max, omega_alpha)
        ts.append(t_next)
        thetas.append(theta_next)
        omegas.append(omega_next)
    omega_theta_oa1.plot(thetas, omegas, linewidth = 1, linestyle =
"None", marker = ".", markersize = 0.2)
    theta_t_oa1.plot(ts, thetas, linewidth = 1, linestyle = "None",
marker = ".", markersize = 0.2)
    omega_t_oa1.plot(ts, omegas, linewidth = 1, linestyle = "None",
marker = ".", markersize = 0.2)
    c += c_step
```

And here is the graph.



Some findings are:

- Pendulums will experience some unstable swinging and then tends to swing along the frequency of the external angular acceleration (ω_α).
- There are shifts between curves of different c. Below some certain ω_α, the peak of smaller c comes earlier than larger c, and vice versa.
- The smaller c is, the larger the amplitude is. (the amplitude during stable swings)
- As ω_α becomes bigger, the amplitude increases at first and reaches highest amplitude at certain ω_α and then decreases.

VI. References

Halliday, D., Resnick, R., & Walker, J. (2021). *Fundamentals of Physics, extended*. John Wiley & Sons.

https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods

https://en.wikipedia.org/wiki/Pendulum_(mechanics)

https://en.wikipedia.org/wiki/Elliptic_integral

https://matplotlib.org/

https://en.wikipedia.org/wiki/Damping