Homework1

SHI Yuting

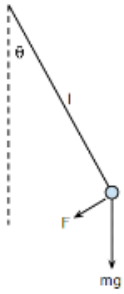
21071210

yshiby@connect.ust.hk

September 18, 2024

1 The linear and nonlinear pendulums

Consider a pendulum with an arm of length l=1.2m holding a bob of mass m. The arm is massless and rigid rod. The gravitational acceleration $g = 9.81m/s^2$. Ignore friction.



1.1

Treat the pendulum as a linear pendulum when the angle is much small. We can use the Newton's second law to get the equation of the pendulum:

$$-mglsin\theta l = m\ddot{\theta}l^2 \tag{1}$$

We can simply the equation to get such like that:

$$\ddot{\theta} = -\frac{gsin\theta}{l} \tag{2}$$

And then we apply the approximation $sin\theta = \theta$, so we could get a formula for simple harmonic motion:

$$\ddot{\theta} = -\frac{g}{l}\theta \tag{3}$$

So we can get the solution:

$$\theta = \theta_0 cos(\sqrt{\frac{g}{l}}t + \phi_0) \tag{4}$$

with θ_0 and ϕ_0 are the constant numbers which will be defined by the initial conditions.

And we could find that the swing period should be:

$$T = 2\pi\sqrt{\frac{l}{g}} \approx 2.198s \tag{5}$$

1.2

I have written a program in Python3.9.13, and is tested that using Python3.10 is also OK. This program is named . You can use

```
python HW1.py
```

to run it. After that you will be asked to enter a initial angle and time step you like.

This program can output a series of pictures, which are comparisons of the results of simulations using the velocity Verlet method and the results of theoretical calculations under the linear model, including comparisons of the angle, angular velocity and energy of the pendulum.

I assumed that the pendulum is released from a standstill as $\theta = 1.9°$, then I set the time step is 0.001s and simulate for 20s. The results are listed.

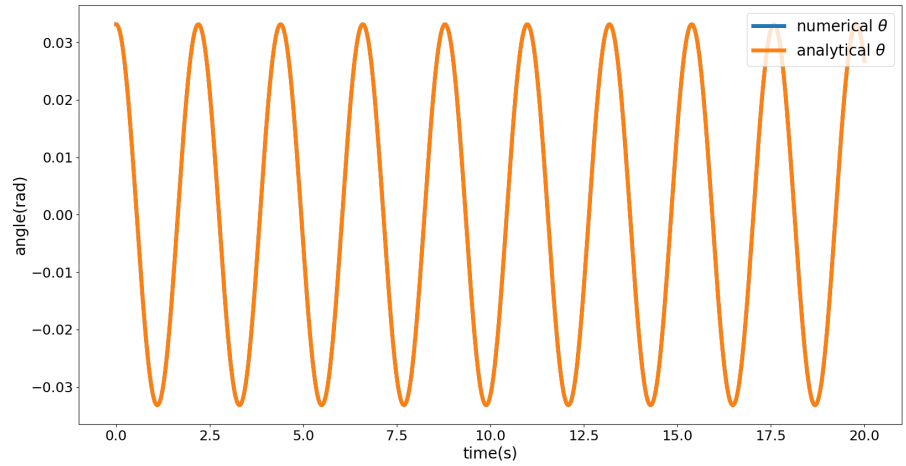


Figure 1: The angle change over time of a pendulum released from 1.9 degrees at a time step of 0.001s

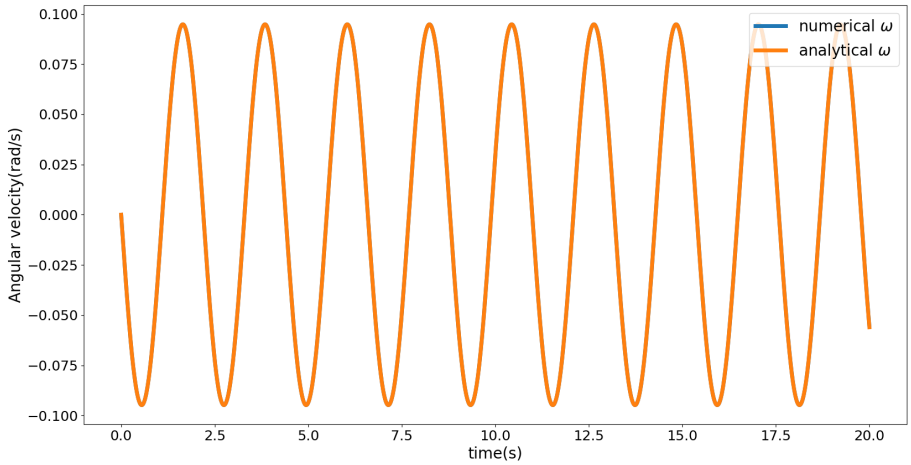


Figure 2: The angular velocity change over time of a pendulum released from 1.9 degrees at a time step of 0.001s

Seems like that there is only one curve in the 1 and 2, the reason is that the 2 curves are too closed to each other. We can see this visually from the figure below.

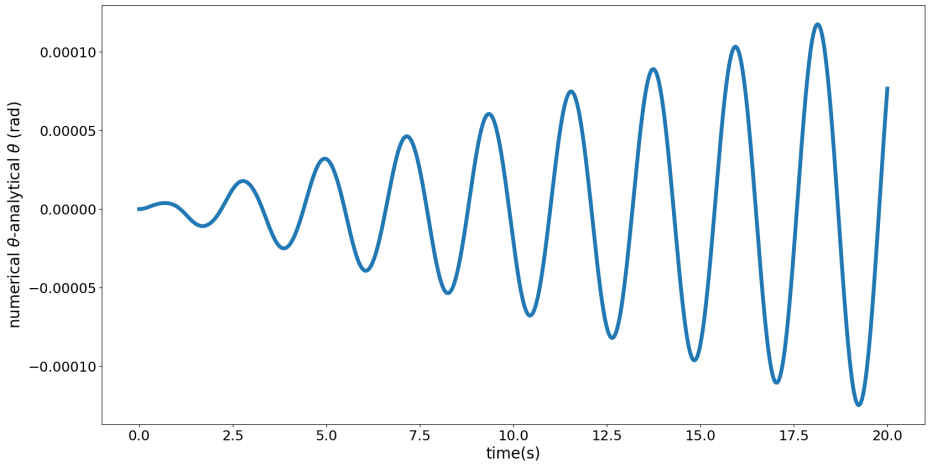


Figure 3: numerical θ-analytical θ during time

We can see, the different between the angle derived by the calculation and the angle derived by the simple harmonic motion equation is pretty small within the 20s we are discussing. However, the different are tending to be larger, so I modified the max time to get the result from 0s to 2000s.

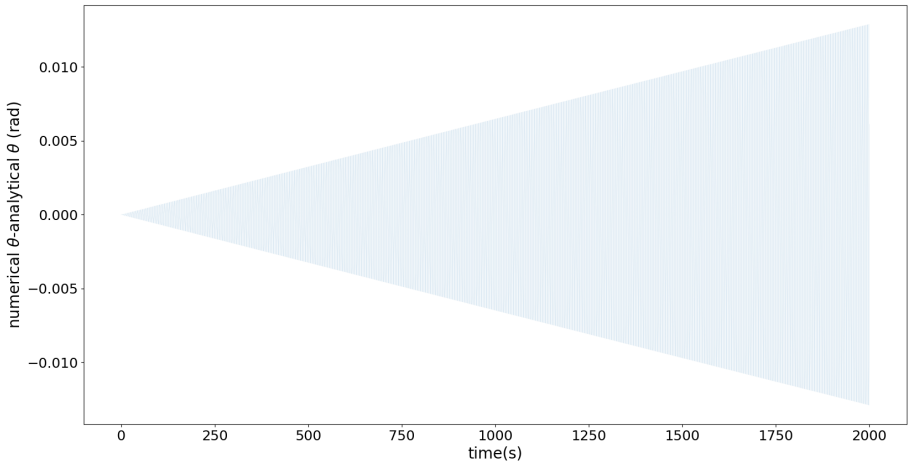


Figure 4: numerical θ-analytical θ during time(2000s)

It seems that there is an envelope for this error, and this error has a certain periodicity, which is similar to half the period of a simple pendulum. This error becomes smaller when the angle reaches the extreme value, and is largest when the angular velocity reaches the extreme value.

In short, after a few hundred seconds, that is, about a few hundred cycles, this difference becomes non-negligible. I think this error can be classified as the accumulation of errors in small-angle approximation, or the loss of accuracy in recursive calculations. It should be noted that when I changed the time step to 0.00001s, the error did not change significantly.

1.3

Considering the nonlinear pendulum released from a standstill at $\theta = 119°$. The total energy consists of two parts, one is the gravity potential energy and the another is the kinetic energy. If we set when $\theta = 0°$ the potential energy is 0 then the energy can be written as:

$$E_p = -mglcos(\theta) + mgl = mgl(1 - cos\theta) \tag{6}$$

$$E_k = \frac{1}{2}ml^2\dot{\theta}^2 \tag{7}$$

So the total energy should be:

$$E = E_k + E_p = mgl(1 - cos\theta) + \frac{1}{2}ml^2\dot{\theta}^2 \tag{8}$$

However, I think that there is no given the number of the mass, if I set the mass=1kg, the total energy $E \approx 17.479J$.

If I use a time step=0.001 and calculate for 40s, I get the result below.

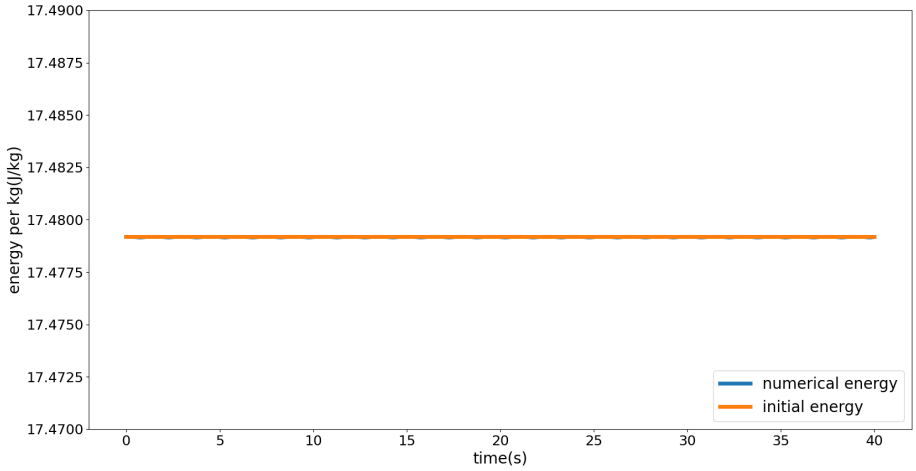


Figure 5: The total energy of the pendulum released from 119 degrees. Calculate 40s using 0.001s as time step.

If you zoom in on the y-axis, you'll see some slight fluctuations. Even so, our energy calculation during these 40 seconds were successful.

Then set the time steps to 0.01s, 0.02s, ..., 0.2s, 0.4s, 0.5s, ..., 0.6s, 0.8s. I finally found that when the time step is 0.51s, significant energy drift starts to occur. Some graphs of energy changes over time at different time steps are shown below.

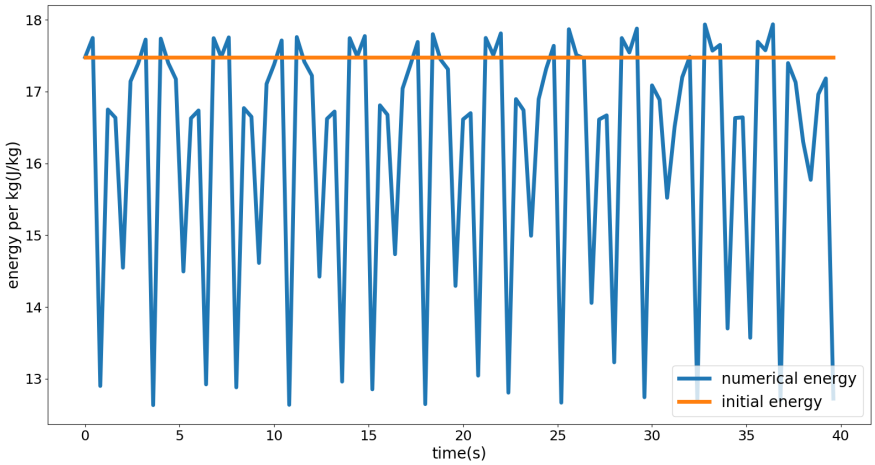
In order to measure the impact of test time on this phenomenon, I set the test time to 2000s. Run respectively on time step of 0.5s and 0.51s. I found that when the time step is 0.5s, although the error of the calculated energy compared to the initial energy is already very large, the error still fluctuates. This shows that it is possible that the phenomenon of energy drift has nothing to do with the running time, which is showed by 7.

Then I focus on the situation that time step = 0.51s. In this case, the angle distribution over time and the energy distribution over time are shown in the 8. From the figure, we can infer that due to the time step being too large, problems occur in the velocity and displacement updates, which in turn leads to greater errors. This can be considered a kind of error accumulation.

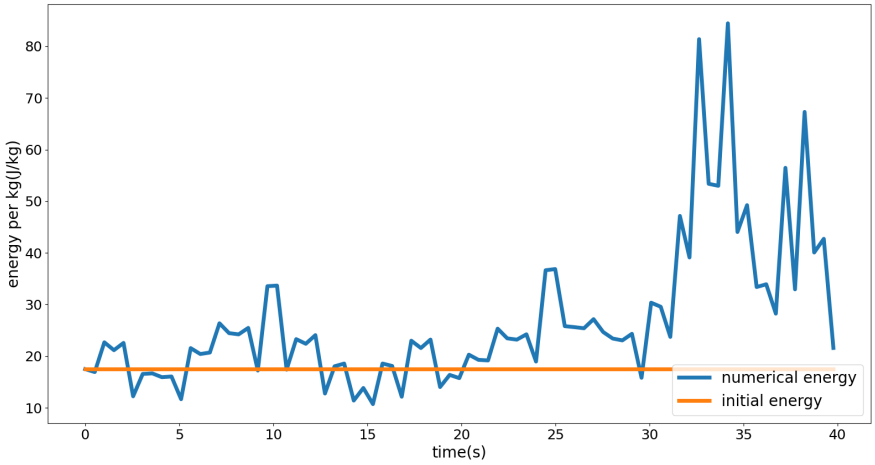
This is obvious because the velocity Verlet method uses the average speed or acceleration over a period of time to represent the instantaneous speed or acceleration, which is only applicable when the time period is particularly small. When the time step is relatively large, it will cause obvious errors and accumulate over time. Especially for a system like a pendulum, when the angle we are calculating

is very close to the amplitude, the speed and acceleration we obtain cannot well represent the situation when the angle of the pendulum reaches its maximum value. We'll get an inappropriate velocity and make our next updated angle exceed the amplitude. These situations exist when the time step is relatively small, and become more significant and exhibit non-periodic behavior when the time step is relatively large. When the time step approaches a quarter cycle, these errors will accumulate in an avalanche manner and cause energy drift.

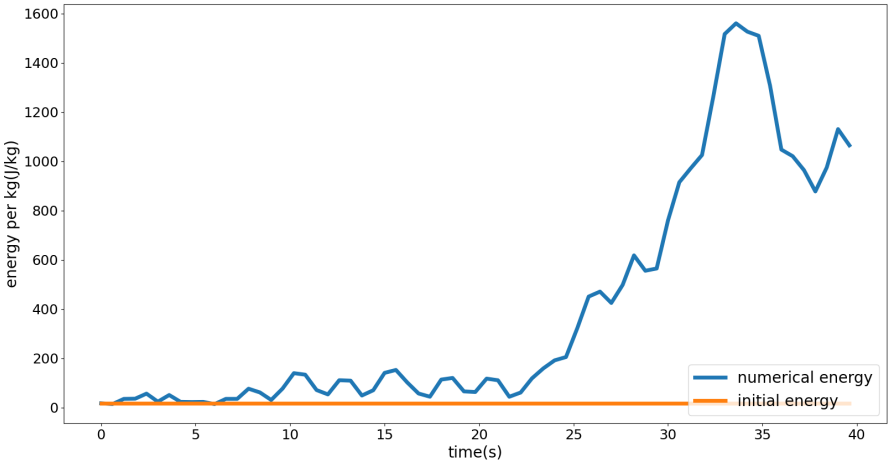
It is worth mentioning that we also found the parametric resonance mentioned by https://en.wikipedia.org/wiki/Energy_drift, which is showed in 6 and 7.



(a) time step=0.4s

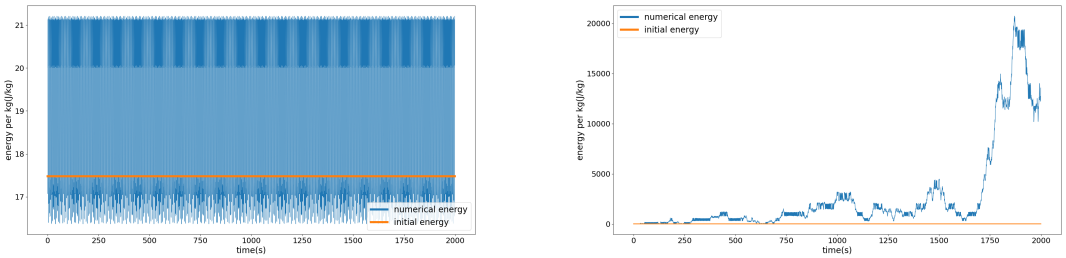


(b) time step=0.51s



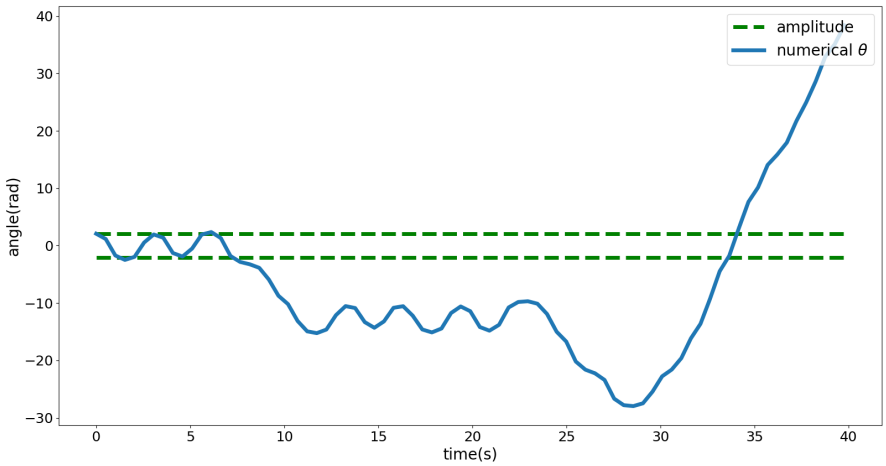
(c) time step=0.6

Figure 6: Graphical representation of energy changes over time at time steps = 0.4s ,0.51s, 0.6s.

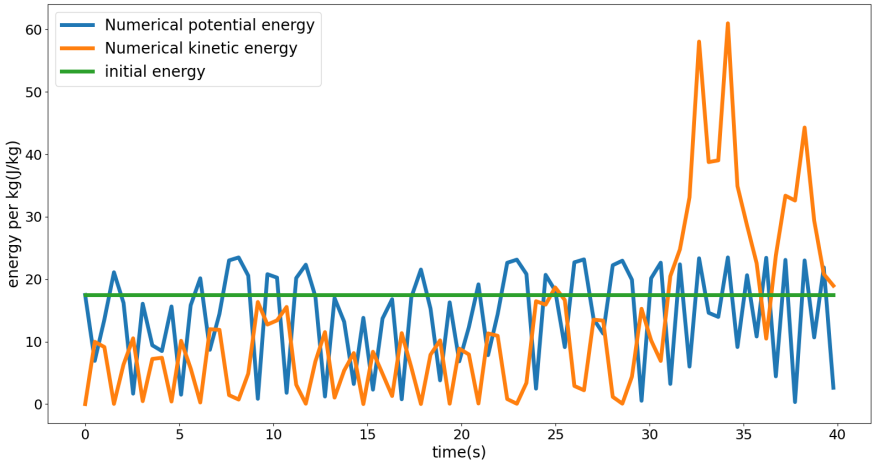


(a) time step=0.5s (b) time step=0.51s

Figure 7: Graphical representation of energy changes over time at time steps = 0.5s, 0.51s, over 2000seconds.



(a) time step=0.5s



(b) time step=0.51s

Figure 8: Graphical representation of angle and energy changes over time at time steps = 0.51s

A

```
import numpy as np
import matplotlib.pyplot as plt
import math
#define the parameter
g=9.81
l=1.2

# Input initial angle in degrees and convert to radians
deg=input("Please enter the initial angle in degrees:")
theta_0=math.radians(float(deg))
omega_0=0

#time step
h=float(input("Please enter the time step in seconds:"))
t_max=40

#initial
time=np.arange(0,t_max,h)
theta=np.zeros(len(time))
omega=np.zeros(len(time))
theta[0]=theta_0
omega[0]=0.0

# Define functions for acceleration, energy, kinetic energy, and
    potential energy
def accelaration(x):
    return -g/l*np.sin(x)
def energy(x,v):
    return 0.5*l*l*v*v+g*l*(1-np.cos(x))
def Kinetic(v):
    return 0.5*l*l*v*v
def V(x):
    return g*l*(1-np.cos(x))

#velocity Verlet
for n in range(1,len(time)):
    theta[n]=theta[n-1]+h*omega[n-1]+0.5*accelaration(theta[n-1])*h*h
    omega[n]=omega[n-1]+0.5*(accelaration(theta[n])+accelaration(theta[n
        -1]))*h

#calculate energy
energy_cal=energy(theta,omega)
Kinetic_num=Kinetic(omega)
V_num=V(theta)

# Calculate analytical solutions for comparison
theta_linear=theta_0*np.cos(np.sqrt(g/l)*time)
omega_linear=-np.sqrt(g/l)*theta_0*np.sin(np.sqrt(g/l)*time)
energy_theory=np.full(len(time),energy(theta[0],omega[0]),dtype=float)
Kinetic_theory=np.full(len(time),Kinetic(omega_linear),dtype=float)
V_theory=np.full(len(time),V(theta_linear),dtype=float)
appli=np.full(len(time),theta_0)

# Plotting the results
```

```
53 plt.xlabel("time(s)",fontsize=20)
54 plt.ylabel("angle(rad)",fontsize=20)
55 plt.xticks(fontsize=18)
56 plt.yticks(fontsize=18)
57 plt.plot(time,appli,linewidth=5,linestyle='—',color='green',label='
       amplitude')
58 plt.plot(time,-appli,linewidth=5,linestyle='—',color='green')
59 plt.plot(time,theta,linewidth=5,label=r'numerical-$\theta$')
60 #plt.plot(time,theta_linear,linewidth=5,label=r"analytical $\theta$")
61 plt.legend(loc=1,fontsize=20)
62 plt.show()
63
64 plt.xlabel("time(s)",fontsize=20)
65 plt.ylabel("Angular-velocity(rad/s)",fontsize=20)
66 plt.plot(time,omega,linewidth=5,label=r'numerical-$\omega$')
67 plt.plot(time,omega_linear,linewidth=5,label=r"analytical-$\omega$")
68 plt.legend(loc=1,fontsize=20)
69 plt.xticks(fontsize=18)
70 plt.yticks(fontsize=18)
71 plt.show()
72
73 plt.xlabel("time(s)",fontsize=20)
74 plt.ylabel(r"numerical-$\theta$-analytical-$\theta$-(rad)",fontsize=20)
75 plt.plot(time,theta-theta_linear,linewidth=5)
76 plt.xticks(fontsize=18)
77 plt.yticks(fontsize=18)
78 plt.show()
79
80 plt.xlabel("time(s)",fontsize=20)
81 plt.ylabel(r"numerical-$\omega$-analytical-$\omega$-(rad/s)",fontsize=20)
82 plt.plot(time,omega-omega_linear,linewidth=5)
83 plt.xticks(fontsize=18)
84 plt.yticks(fontsize=18)
85 plt.show()
86
87 plt.ylabel("energy-per-kg(J/kg)",fontsize=20)
88 plt.xlabel("time(s)",fontsize=20)
89 plt.plot(time,energy_cal,linewidth=5,label="numerical-energy")
90 plt.plot(time,energy_theory,linewidth=5,label="initial-energy")
91 plt.legend(loc=2,fontsize=20)
92 plt.xticks(fontsize=18)
93 plt.yticks(fontsize=18)
94 #plt.ylim(17.47,17.49)
95 plt.show()
96
97 plt.ylabel("kinetic-energy-per-kg(J/kg)",fontsize=20)
98 plt.xlabel("time(s)",fontsize=20)
99 plt.plot(time,energy_theory,linewidth=5,label="initial-energy")
100 plt.plot(time,Kinetic_num,linewidth=5,label="Numerical-kinetic-energy")
101 #plt.plot(time,Kinetic_theory,linewidth=5,label="Analytical kinetic
       energy")
102 plt.legend(loc=2,fontsize=20)
103 plt.xticks(fontsize=18)
104 plt.yticks(fontsize=18)
105
106 plt.show()
```

```
107
108 plt.ylabel("potential energy per kg(J/kg)",fontsize=20)
109 plt.xlabel("time(s)",fontsize=20)
110 plt.plot(time,energy_theory,linewidth=5,label="initial energy")
111 plt.plot(time,V_num,linewidth=5,label="Numerical potential energy")
112 #plt.plot(time,V_theory,linewidth=5,label="Analytical potential energy")
113 plt.legend(loc=2,fontsize=20)
114 plt.xticks(fontsize=18)
115 plt.yticks(fontsize=18)
116
117 plt.show()
118
119 plt.ylabel("energy per kg(J/kg)",fontsize=20)
120 plt.xlabel("time(s)",fontsize=20)
121 plt.plot(time,V_num,linewidth=5,label="Numerical potential energy")
122 plt.plot(time,Kinetic_num,linewidth=5,label="Numerical kinetic energy")
123 plt.plot(time,energy_theory,linewidth=5,label="initial energy")
124 plt.legend(loc=2,fontsize=20)
125 plt.xticks(fontsize=18)
126 plt.yticks(fontsize=18)
127
128 plt.show()
129 # Print final values of theta and omega
130 #print(theta)
131 #print(omega)
```